

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
Современные технологии программирования

Практическая работа №10
«Абстрактный тип данных (ADT) «Множество»»

Выполнил: студент 4 курса
группы ИП-111
Кузьменок Денис Витальевич

Проверил преподаватель:
Зайцев Михаил Георгиевич

Новосибирск, 2024 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 Visual Studio.

3. Если необходимо, предусмотрите возбуждение исключительных ситуаций.

Спецификация абстрактного 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.

Таблица 1. Описание операций на ADT tset.

Наименование Операции	Описание
Конструктор	
Начальные	Нет.
значения:	
Процесс:	Создаёт пустое множество элементов типа T.
Опустошить	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Удаляет из множества все элементы.
Выход:	Нет.
Постусловия:	Множество - пусто.
Добавить	
Вход:	d – элемент типа T.
Предусловия:	Нет.
Процесс:	Добавляет d во множество, если в нем нет такого элемента.
Выход:	Нет.
Постусловия:	Множество содержит элемент d.
Удалить	
Вход:	d – элемент типа T.
Предусловия:	Нет.
Процесс:	Удаляет элемент d из множества, если d принадлежит множеству.
Выход:	Нет.
Постусловия:	Множество не содержит элемент d.

<i>Пусто</i>	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Определяет, содержит ли множество элементы. Возвращает значение True, если множество не пусто, False – в противном случае.
Выход:	Булевское значение.
Постусловия:	Нет.
<i>Принадлежит</i>	
Вход:	d – элемент типа T.
Предусловия:	Нет.
Процесс:	Определяет, принадлежит ли элемент d множеству. Возвращает True, если d принадлежит множеству, False – в противном случае.

Выход:	Булевское значение.
Постусловия:	Нет.
<i>Объединить</i>	
Вход:	Множество q.
Предусловия:	Нет.
Процесс:	Создаёт множество, полученное в результате объединения множества с множеством q.
Выход:	Множество.
Постусловия:	Нет.
<i>Вычесть</i>	
Вход:	Множество q.
Предусловия:	Нет.
Процесс:	Создаёт множество, полученное в результате вычитания из множества множество q.
Выход:	Множество.
Постусловия:	Нет.
<i>Умножить</i>	
Вход:	Множество q.
Предусловия:	Нет.
Процесс:	Создаёт множество, являющееся пересечением множества с множеством q.
Выход:	Множество.
Постусловия:	Нет.

Элементов	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Подсчитывает и возвращает количество элементов во множестве, если множество пустое - ноль
Выход:	Целое - количество элементов во множестве.
Постусловия:	Нет.
Элемент	
Вход:	j - номер элемента множества.
Предусловия:	Нет.
Процесс:	Обеспечивает доступ к элементу множества для чтения по индексу j так, что если
	изменять j от 1 до количества элементов во множестве, то можно просмотреть все элементы множества.
Выход:	Элемент множества типа T.
Постусловия:	Множество не модифицируется

Реализация:

```
Множество set1 = { 1 3 6 }
Множество set2 = { 2 1 5 }
Множество set3 = { 87 -2 0 -256 43 1 }
Множество set1 без элементов, входящих в множество set2 = { 3 6 }
Объединение множеств set1 и set2 = { 1 3 6 2 5 }
Пересечение множеств set1 и set2 = { 1 }

Удаление элемента -256 из множества set3 = { 87 -2 0 43 1 }
Очищено ли множество set3? Нет
Есть в множестве set3 4? Нет
А элемент 43? Да
Количество элементов в множестве set3 = 5
Элемент на позиции 3 = 0
А сейчас очищено ли множество set3? Да
```

Рис. 1 – Результат проверки работоспособности программы.

Обозреватель тестов

20 20 0

Обнаружение тестов пропущено. Все контейнеры тестов обновлены

Тестирование	Длительность	Признаки	Сообщение об ошибке
SetTest (20)	44 мс		
SetTest (20)	44 мс		
SetTests (20)	44 мс		
Copy	1 мс		
Equals	< 1 мс		
NotEqualsDi...	< 1 мс		
NotEqualsDi...	< 1 мс		
TestClear	< 1 мс		
TestContains	< 1 мс		
TestContains2	< 1 мс		
TestCount	1 мс		
TestCountWi...	< 1 мс		
TestElement...	< 1 мс		
TestElement...	18 мс		
TestExcept	< 1 мс		
TestExceptW...	1 мс		
TestIntersect	< 1 мс		
TestIntersect...	23 мс		
TestIsClear	< 1 мс		
TestIsClear2	< 1 мс		
TestRemove	< 1 мс		
TestUnion	< 1 мс		
TestUnionWi...	< 1 мс		

Выполнить | Отладка

Сводка по группе

SetTest

Тесты в группе: 20

Общая длительность: 44 мс

Результаты

20 Пройден

Рис. 2 – Результат выполнения модульных тестов.

1. TestElementAt1

- **Что проверяет:** корректность возврата элемента из множества по индексу.
- **Входные значения:**
 - `TSet<int> set1 = new TSet<int>()` — создаётся объект класса `TSet`.
 - `set1.Add(1);`
 - `set1.Add(2);`
 - `set1.Add(4);`
- **Ожидаемое значение:** “1, 2, 4” – элементы по индексам 0, 1, 2.

2. TestElementAt2

- **Что проверяет:** корректность возврата элемента из множества по индексу.
- **Входные значения:**
 - `TSet<int> set1 = new TSet<int>()` — создаётся объект класса `TSet`.
 - `set1.Add(1);`
 - `set1.Add(2);`
 - `set1.Add(4);`
- **Ожидаемое значение:** генерация исключения `ArgumentOutOfRangeException` – т.к. нельзя взять элемент по индексу 5.

3. TestClear

- **Что проверяет:** корректность удаления всех элементов из множества.
- **Входные значения:**
 - `TSet<int> set1 = new TSet<int>()` — создаётся объект класса `TSet`.
 - `set1.Add(2);`

- `set1.Add(3);`
- `set1.Add(4);`
- `set1.Clear();`
- **Ожидаемое значение:** “{ }” – обозначение пустого множества.

4. TestRemove

- **Что проверяет:** корректность удаления элемента из множества.
- **Входные значения:**
 - `TSet<int> set1 = new TSet<int>()` — создаётся объект класса `TSet`.
 - `set1.Add(-11);`
 - `set1.Add(8);`
 - `set1.Add(62);`
 - `set1.Remove(8);`
- **Ожидаемое значение:** “{ -11, 62 }” – множество с удалённым элементом 8.

5. TestIsClear

- **Что проверяет:** проверка множества на пустоту.
- **Входные значения:**
 - `TSet<int> set1 = new TSet<int>()` — создаётся объект класса `TSet`.
 - `set1.Add(22);`
 - `set1.Add(0);`
 - `set1.Add(4);`
 - `Assert.IsFalse(set1.IsClear());`
- **Ожидаемое значение:** “false” – т.к. множество содержит в себе элементы.

6. TestIsClear2

- **Что проверяет:** проверка множества на пустоту.
- **Входные значения:**
 - `TSet<int> set1 = new TSet<int>()` — создаётся объект класса `TSet`.
 - `set1.Add(91);`
 - `set1.Add(-621);`
 - `set1.Add(82);`
 - `set1.Clear();`
 - `Assert.IsTrue(set1.IsClear());`
- **Ожидаемое значение:** “true” — т.к. множество не содержит в себе элементов.

7. TestContains

- **Что проверяет:** проверка существования элемента в текущем множестве.
- **Входные значения:**
 - `TSet<int> set1 = new TSet<int>()` — создаётся объект класса `TSet`.
 - `set1.Add(9);`
 - `set1.Add(1);`
 - `set1.Add(36);`
 - `Assert.IsTrue(set1.Contains(36));`
- **Ожидаемое значения:** “true” — т.к. элемент 36 существует в текущем множестве.

8. TestContains2

- **Что проверяет:** проверка существования элемента в текущем множестве.

- **Входные значения:**

- `TSet<int> set1 = new TSet<int>()` — создаётся объект класса `TSet`.
- `set1.Add(-5);`
- `set1.Add(5);`
- `set1.Add(-873);`
- `Assert.IsFalse(set1.Contains(-874));`

- **Ожидаемое значения:** “false” – т.к. элемента -874 нет в множестве.

9. TestUnion

- **Что проверяет:** корректность объединения двух множеств.

- **Входные значения:**

- `TSet<int> set1 = new TSet<int>()` — создаётся объект класса `TSet`.
- `set1.Add(4);`
- `set1.Add(-9);`
- `set1.Add(77);`
- `TSet<int> set2 = new TSet<int>()` — создаётся второй объект класса `TSet`.
- `set2.Add(6);`
- `set2.Add(-9);`
- `set2.Add(4);`

- **Ожидаемое значения:** “{ 4, -9, 77, 6 }” – результат объединения двух множеств без повторения элементов.

10. TestUnionWithEmptySet

- **Что проверяет:** корректность объединения с пустым множеством.

- **Входные значения:**

- `TSet<int> set1 = new TSet<int>()` — создаётся объект класса `TSet`.
- `set1.Add(4);`
- `set1.Add(-9);`
- `set1.Add(77);`
- `TSet<int> set2 = new TSet<int>()` — создаётся второй объект класса `TSet`.

- **Ожидаемое значения:** “{ 4, -9, 77 }” – результат объединения первого множества со вторым пустым.

11. TestExcept

- **Что проверяет:** корректность вычитания второго множества из первого.

- **Входные значения:**

- `TSet<int> set1 = new TSet<int>()` — создаётся объект класса `TSet`.
- `set1.Add(83);`
- `set1.Add(3);`
- `set1.Add(2);`
- `TSet<int> set2 = new TSet<int>()` — создаётся второй объект класса `TSet`.
- `set2.Add(3);`
- `set2.Add(97);`
- `set2.Add(2);`

- **Ожидаемое значения:** “{ 83 }” – т.к. это единственный элемент из первого множества, не встречающийся во втором.

12. TestExceptWithAllSameElements

- **Что проверяет:** корректность вычитания второго множества из первого при условии, что все элементы из двух множеств совпадают.
- **Входные значения:**
 - `TSet<int> set1 = new TSet<int>()` — создаётся объект класса `TSet`.
 - `set1.Add(97);`
 - `set1.Add(3);`
 - `set1.Add(2);`
 - `TSet<int> set2 = new TSet<int>()` — создаётся второй объект класса `TSet`.
 - `set2.Add(3);`
 - `set2.Add(97);`
 - `set2.Add(2);`
- **Ожидаемое значения:** “{ }” — пустое множество, т.к. все элементы из первого множества встречаются во втором.

13. TestIntersect

- **Что проверяет:** корректность пересечения двух множеств.
- **Входные значения:**
 - `TSet<int> set1 = new TSet<int>()` — создаётся объект класса `TSet`.
 - `set1.Add(9);`
 - `set1.Add(0);`
 - `set1.Add(-53);`
 - `TSet<int> set2 = new TSet<int>()` — создаётся второй объект класса `TSet`.

- `set2.Add(0);`
- `set2.Add(8);`
- `set2.Add(72);`
- **Ожидаемое значения:** “{ 0 }” — т.к. это единственный элемент, который принадлежит обоим множествам.

14. **TestIntersectNoSameElements**

- **Что проверяет:** корректность пересечения двух множеств.
- **Входные значения:**
 - `TSet<int> set1 = new TSet<int>()` — создаётся объект класса `TSet`.
 - `set1.Add(9);`
 - `set1.Add(0);`
 - `set1.Add(-53);`
 - `TSet<int> set2 = new TSet<int>()` — создаётся второй объект класса `TSet`.
 - `set2.Add(2);`
 - `set2.Add(8);`
 - `set2.Add(72);`
- **Ожидаемое значения:** “{ }” — т.к. нет элементов, которые есть и в первом множестве, и во втором.

15. **TestCount**

- **Что проверяет:** количество элементов в множестве.
- **Входные значения:**
 - `TSet<int> set1 = new TSet<int>()` — создаётся объект класса `TSet`.
 - `set1.Add(10);`
 - `set1.Add(2);`

- `set1.Add(-4);`

- **Ожидаемое значения:** “3” — уникальных элементов в множестве.

16. **TestCountWithSameElements**

- **Что проверяет:** количество элементов в множестве.

- **Входные значения:**

- `TSet<int> set1 = new TSet<int>()` — создаётся объект класса `TSet`.

- `set1.Add(10);`

- `set1.Add(2);`

- `set1.Add(2);`

- **Ожидаемое значения:** “2” — уникальных элементов в множестве.

Вывод

В результате работы над лабораторной работой были сформированы практические навыки реализации параметризованного абстрактного типа данных на языке C#, разработки функций классов на языке C#, разработка модульных тестов для тестирования функций классов и выполнения модульного тестирования на языке C# с помощью средств автоматизации Visual Studio.

Листинг программы:

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace lab10
{
    class Program
    {
        static void Main(string[] args)
        {
            TSet<int> set1 = new TSet<int>();
            TSet<int> set2 = new TSet<int>();
            TSet<int> set3 = new TSet<int>();
            set1.Add(1);
            set1.Add(3);
            set1.Add(6);

            set2.Add(2);
            set2.Add(1);
            set2.Add(5);

            set3.Add(87);
            set3.Add(-2);
            set3.Add(0);
            set3.Add(-256);
            set3.Add(43);
            set3.Add(1);

            Console.WriteLine($"Множество set1 = {set1.Show()}");
            Console.WriteLine($"Множество set2 = {set2.Show()}");
            Console.WriteLine($"Множество set3 = {set3.Show()}");

            Console.WriteLine($"Множество set1 без элементов, входящих в
множество set2 = {set1.Except(set2).Show()}");
            Console.WriteLine($"Объединение множеств set1 и set2 =
{set1.Union(set2).Show()}");
            Console.WriteLine($"Пересечение множеств set1 и set2 =
{set1.Intersect(set2).Show()}");
            Console.WriteLine();

            set3.Remove(-256);
            Console.WriteLine($"Удаление элемента -256 из множества set3 =
{set3.Show()}");
            Console.WriteLine($"Очищено ли множество set3? {(set3.IsClear() ?
"Да" : "Нет")}");
            Console.WriteLine($"Есть в множестве set3 4? {(set3.Contains(4) ?
"Да" : "Нет")}");
            Console.WriteLine($"А элемент 43? {(set3.Contains(43) ? "Да" :
"Нет")}");

            int count = set3.Count();
            Console.WriteLine($"Количество элементов в множестве set3 =
{count}");
        }
    }
}
```



```
        int position = set3.ElementAt(2);  
        Console.WriteLine($"Элемент на позиции 3 = {position}");  
        set3.Clear();  
        Console.WriteLine($"А сейчас очищено ли множество set3?  
{(set3.IsClear() ? "Да" : "Нет")}");  
    }  
}
```

TSet.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace lab10
{
    public class TSet<T> where T : new()
    {
        public HashSet<T> LocalSet;

        public TSet()
        {
            LocalSet = new HashSet<T>();
        }

        public void Clear()
        {
            LocalSet.Clear();
        }

        public void Add(T element)
        {
            LocalSet.Add(element);
        }

        public void Remove(T element)
        {
            LocalSet.Remove(element);
        }

        public bool IsClear()
        {
            return true ? LocalSet.Count == 0 : false;
        }

        public bool Contains(T element)
        {
            return LocalSet.Contains(element);
        }

        public int Count()
        {
            return this.LocalSet.Count();
        }

        public T ElementAt(int index)
        {
            try
            {
                object needed = this.LocalSet.ElementAt(index);
                return (T)needed;
            }
            catch
            {
            }
        }
    }
}
```

```

        throw new ArgumentOutOfRangeException();
    }
}

public String Show()
{
    StringBuilder sb = new StringBuilder();

    sb.Append("{ ");

    foreach (T element in this.LocalSet)
    {
        sb.Append($"{element.GetType().GetMethod("Show").Invoke(element, null) ??
element} ");
    }
    sb.Append("}");
    return sb.ToString();
}

public TSet<T> Union(TSet<T> other)
{
    TSet<T> newSet = new TSet<T>();

    foreach (T element in this.LocalSet)
    {
        newSet.Add(element);
    }

    foreach (T element in other.LocalSet)
    {
        newSet.Add(element);
    }

    return newSet;
}

public TSet<T> Except(TSet<T> other)
{
    TSet<T> newSet = new TSet<T>();

    foreach (T element in this.LocalSet)
    {
        newSet.Add(element);
    }

    foreach (T element in other.LocalSet)
    {
        newSet.Remove(element);
    }

    return newSet;
}

public TSet<T> Intersect(TSet<T> other)
{
    TSet<T> newSet = new TSet<T>();

    foreach (T element in this.LocalSet)
    {

```

```

        if (other.Contains(element))
        {
            newSet.Add(element);
        }
    }

    return newSet;
}

public override bool Equals(object obj)
{
    if (!(obj is TSet<T> otherSet))
    {
        return false;
    }

    if (this.LocalSet.Count != otherSet.LocalSet.Count)
    {
        return false;
    }

    foreach (T item in this.LocalSet)
    {
        if (!otherSet.LocalSet.Contains(item))
        {
            return false;
        }
    }

    return true;
}

public TSet<T> Copy()
{
    return (TSet<T>) this.MemberwiseClone();
}
}

```

UnitTests1.cs

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System;
using lab10;

namespace SetTest
{
    [TestClass]
    public class SetTests
    {
        [TestMethod]
        public void TestElementAt1()
        {
            TSet<int> set1 = new TSet<int>();
            set1.Add(1);
            set1.Add(2);
            set1.Add(4);
            Assert.AreEqual(set1.ElementAt(0), 1);
            Assert.AreEqual(set1.ElementAt(1), 2);
            Assert.AreEqual(set1.ElementAt(2), 4);
        }

        [TestMethod]
        [ExpectedException(typeof(ArgumentOutOfRangeException))]
        public void TestElementAt2()
        {
            TSet<int> set1 = new TSet<int>();
            set1.Add(1);
            set1.Add(2);
            set1.Add(4);
            Assert.AreEqual(set1.ElementAt(0), 1);
            Assert.AreEqual(set1.ElementAt(1), 2);
            Assert.AreEqual(set1.ElementAt(5), 4);
        }

        [TestMethod]
        public void TestClear()
        {
            TSet<int> set1 = new TSet<int>();
            set1.Add(2);
            set1.Add(3);
            set1.Add(4);

            string actual = "{ 2 3 4 }";

            Assert.AreEqual(set1.Show(), actual);

            set1.Clear();
            actual = "{ }";
            Assert.AreEqual(set1.Show(), actual);
        }

        [TestMethod]
        public void TestRemove()
        {
            TSet<int> set1 = new TSet<int>();
            set1.Add(-11);
            set1.Add(8);
            set1.Add(62);
        }
    }
}
```

```

        set1.Remove(8);

        string actual = "{ -11 62 }";

        Assert.AreEqual(set1.Show(), actual);
    }

    [TestMethod]
    public void TestIsClear()
    {
        TSet<int> set1 = new TSet<int>();
        set1.Add(22);
        set1.Add(0);
        set1.Add(4);

        Assert.IsFalse(set1.IsClear());
    }

    [TestMethod]
    public void TestIsClear2()
    {
        TSet<int> set1 = new TSet<int>();
        set1.Add(91);
        set1.Add(-621);
        set1.Add(82);
        set1.Clear();

        Assert.IsTrue(set1.IsClear());
    }

    [TestMethod]
    public void TestContains()
    {
        TSet<int> set1 = new TSet<int>();
        set1.Add(9);
        set1.Add(1);
        set1.Add(36);

        Assert.IsTrue(set1.Contains(36));
    }

    [TestMethod]
    public void TestContains2()
    {
        TSet<int> set1 = new TSet<int>();
        set1.Add(-5);
        set1.Add(5);
        set1.Add(-873);

        Assert.IsFalse(set1.Contains(-874));
    }

    [TestMethod]
    public void TestUnion()
    {
        TSet<int> set1 = new TSet<int>();
        set1.Add(4);
        set1.Add(-9);
        set1.Add(77);
    }

```

```

TSet<int> set2 = new TSet<int>();
set2.Add(6);
set2.Add(-9);
set2.Add(4);

string actual = "{ 4 -9 77 6 }";

Assert.AreEqual(set1.Union(set2).Show(), actual);
}

[TestMethod]
public void TestUnionWithEmptySet()
{
    TSet<int> set1 = new TSet<int>();
    set1.Add(4);
    set1.Add(-9);
    set1.Add(77);

    TSet<int> set2 = new TSet<int>();

    string actual = "{ 4 -9 77 }";

    Assert.AreEqual(set1.Union(set2).Show(), actual);
}

[TestMethod]
public void TestExcept()
{
    TSet<int> set1 = new TSet<int>();
    set1.Add(83);
    set1.Add(3);
    set1.Add(2);

    TSet<int> set2 = new TSet<int>();
    set2.Add(3);
    set2.Add(97);
    set2.Add(2);

    string actual = "{ 83 }";

    Assert.AreEqual(set1.Except(set2).Show(), actual);
}

[TestMethod]
public void TestExceptWithAllSameElements()
{
    TSet<int> set1 = new TSet<int>();
    set1.Add(97);
    set1.Add(3);
    set1.Add(2);

    TSet<int> set2 = new TSet<int>();
    set2.Add(3);
    set2.Add(97);
    set2.Add(2);

    string actual = "{ }";

```

```

        Assert.AreEqual(set1.Except(set2).Show(), actual);
    }

    [TestMethod]
    public void TestIntersect()
    {
        TSet<int> set1 = new TSet<int>();
        set1.Add(9);
        set1.Add(0);
        set1.Add(-53);

        TSet<int> set2 = new TSet<int>();
        set2.Add(0);
        set2.Add(8);
        set2.Add(72);

        string actual = "{ 0 }";

        Assert.AreEqual(set1.Intersect(set2).Show(), actual);
    }

    [TestMethod]
    public void TestIntersectNoSameElements()
    {
        TSet<int> set1 = new TSet<int>();
        set1.Add(9);
        set1.Add(0);
        set1.Add(-53);

        TSet<int> set2 = new TSet<int>();
        set2.Add(2);
        set2.Add(8);
        set2.Add(72);

        string actual = "{ }";

        Assert.AreEqual(set1.Intersect(set2).Show(), actual);
    }

    [TestMethod]
    public void TestCount()
    {
        TSet<int> set1 = new TSet<int>();
        set1.Add(10);
        set1.Add(2);
        set1.Add(-4);

        int actual = 3;

        Assert.AreEqual(set1.Count(), actual);
    }

    [TestMethod]
    public void TestCountWithSameElements()
    {
        TSet<int> set1 = new TSet<int>();
        set1.Add(10);
        set1.Add(2);
        set1.Add(2);
    }

```



```

        int actual = 2;

        Assert.AreEqual(set1.Count(), actual);
    }

    [TestMethod]
    public void Equals()
    {
        TSet<int> set1 = new TSet<int>();
        TSet<int> set2 = new TSet<int>();

        set1.Add(10);
        set1.Add(2);
        set1.Add(-7);
        set2.Add(-7);
        set2.Add(10);
        set2.Add(2);

        Assert.IsTrue(set1.Equals(set2));
    }

    [TestMethod]
    public void NotEqualsDifferentCount()
    {
        TSet<int> set1 = new TSet<int>();
        TSet<int> set2 = new TSet<int>();

        set1.Add(10);
        set1.Add(-7);
        set2.Add(-7);
        set2.Add(10);
        set2.Add(2);

        Assert.IsFalse(set1.Equals(set2));
    }

    [TestMethod]
    public void NotEqualsDifferentElements()
    {
        TSet<int> set1 = new TSet<int>();
        TSet<int> set2 = new TSet<int>();

        set1.Add(10);
        set1.Add(-7);
        set1.Add(3);
        set2.Add(-7);
        set2.Add(10);
        set2.Add(2);

        Assert.IsFalse(set1.Equals(set2));
    }

    [TestMethod]
    public void Copy()
    {
        TSet<int> set1 = new TSet<int>();
        TSet<int> set2 = new TSet<int>();
    }

```

```
        set1.Add(4);  
        set1.Add(0);  
        set1.Add(-64);  
        set1.Add(72);  
  
        set2 = set1.Copy();  
  
        Assert.IsTrue(set1.Equals(set2));  
    }  
}
```