Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

**МЕХАНІКО-ТЕХНОЛОГІЧНИЙ ФАКУЛЬТЕТ**

Кафедра кібербезпеки та програмного забезпечення

**Звіт**

з виконаної лабораторної роботи № 1

Дисципліна: Системне програмне забезпечення

Виконав :

студент академічної групи КІ-15

Аннаєв А. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Перевірив :

Викладач

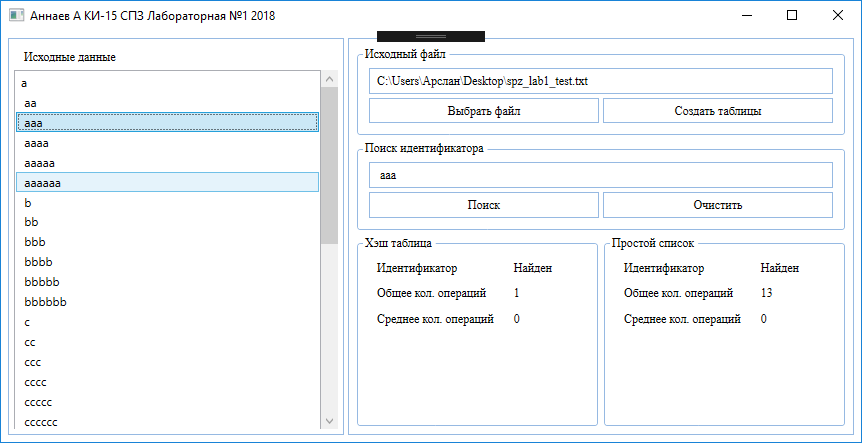
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кропивницкий- 2018

Лабораторная работа № 1

**Тема**: Организация таблиц идентификаторов.

**Цель**: Изучить основные методы организации таблиц идентификаторов, получить представление о преимуществах и недостатках, присущих разным методам организации таблиц идентификаторов.



using CompilerAPI.Enum;

namespace CompilerAPI.LexemItentifierModel

{

/// <summary>

/// Структура идентификатора

/// </summary>

public class IdentitifierInfo

{

public string ItemName { get; private set; }

public object ItemValue { get; set; }

public IdentifierType ItemType {get; private set;}

public IdentitifierInfo(string key, IdentifierType type)

{

ItemName = key;

ItemType = type;

}

}

}

using CompilerAPI.Helper;

using CompilerAPI.Interface;

using System;

using System.Collections;

using System.Collections.ObjectModel;

using System.Diagnostics;

using System.Linq;

namespace CompilerAPI.LexemItentifierModel

{

/// <summary>

/// Таблица идентификаторов построенная на основе хэш таблиц

/// </summary>

public class HashIdentityTable : BindingProperty, IIdentityFeature

{

#region Переменные

/// <summary>

/// Максимальная длина ключа

/// </summary>

public const uint MAX\_IDENTIFY\_LENGTH = 32;

/// <summary>

/// Внутреннее представление таблицы

/// </summary>

private Hashtable table;

/// <summary>

/// Диагностический таймеры

/// </summary>

private readonly Stopwatch timer;

#endregion

#region Свойства

/// <summary>

/// Таблица элементов, только для чтения

/// </summary>

public ReadOnlyCollection<IdentitifierInfo> Items

{

get

{ return table.Values.Cast<IdentitifierInfo>().ToList().AsReadOnly();}

}

/// <summary>

/// Общее количество операций

/// </summary>

private int \_totalOperationCount;

public int TotalOperationCount

{

get { return \_totalOperationCount; }

set

{

\_totalOperationCount = value;

OnPropertyChanged(nameof(TotalOperationCount));

}

}

/// <summary>

/// Среднее количество операций

/// </summary>

private int \_middleOperationCount;

public int MiddleOperationCount

{

get { return \_middleOperationCount; }

set

{

\_middleOperationCount = value;

OnPropertyChanged(nameof(MiddleOperationCount));

}

}

/// <summary>

/// Время затраченное на выполнение одной из операций

/// </summary>

private string \_elapseTime;

public string ElapsedTime

{

get { return \_elapseTime; }

set

{

\_elapseTime = value;

OnPropertyChanged(nameof(ElapsedTime));

}

}

#endregion

#region Конструктор

/// <summary>

/// Конструктор по умолчанию

/// </summary>

public HashIdentityTable()

{

table = new Hashtable();

timer = new Stopwatch();

}

#endregion

#region Реализация интерфейса

/// <summary>

/// Добавление элемента в таблицу

/// </summary>

/// <param name="value"></param>

public void AddItem(IdentitifierInfo value)

{

if (value == null)

{ throw new ArgumentNullException(nameof(value)); }

if (string.IsNullOrEmpty(value.ItemName))

{ throw new ArgumentNullException(nameof(value.ItemName)); }

if (value.ItemName.Length > MAX\_IDENTIFY\_LENGTH)

{ throw new ArgumentNullException(@"Длина ключа превышает лимит в

{MAX\_IDENTIFY\_LENGTH}"); }

if (table.ContainsKey(value.ItemName))

{ throw new Exception("Элемент с таким ключом присутствует" +

value.ItemName); }

table.Add(value.ItemName, value);

TotalOperationCount += 1;

}

/// <summary>

/// Поиск элемента в таблице по ключу

/// </summary>

/// <param name="key">ключ</param>

/// <returns>Найденный объект или null</returns>

public IdentitifierInfo FindItem(string key)

{

if (string.IsNullOrEmpty(key))

{ throw new ArgumentNullException(nameof(key)); }

if (table == null)

{ throw new ArgumentNullException(nameof(table)); }

if (!table.ContainsKey(key))

{ return null; }

IdentitifierInfo item = table[key] as IdentitifierInfo;

TotalOperationCount += 1;

return item;

}

/// <summary>

/// Удаление элемента по ключу

/// </summary>

/// <param name="key"></param>

/// <returns></returns>

public bool RemoveItem(string key)

{

if (string.IsNullOrEmpty(key))

{ throw new ArgumentNullException(nameof(key)); }

if (table == null)

{ throw new ArgumentNullException(nameof(table)); }

if (!table.ContainsKey(key))

{ return false; }

table.Remove(key);

TotalOperationCount += 1;

return false;

}

/// <summary>

/// Очистка всей таблицы

/// </summary>

public void ClearItems()

{

TotalOperationCount = 0;

if (table != null)

{

table.Clear();

}

}

#endregion

}

}

using CompilerAPI.Helper;

using CompilerAPI.Interface;

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Diagnostics;

namespace CompilerAPI.LexemItentifierModel

{

/// <summary>

/// Таблица идентификаторов

/// </summary>

public class SimpleIndentityTable : BindingProperty, IIdentityFeature

{

#region Переменные

/// <summary>

/// Максимальная длина ключа

/// </summary>

public const uint MAX\_IDENTIFY\_LENGTH = 32;

/// <summary>

/// Внутреннее представление таблицы

/// </summary>

private List<IdentitifierInfo> \_items;

/// <summary>

/// Диагностический таймеры

/// </summary>

private readonly Stopwatch timer;

#endregion

#region Свойства

/// <summary>

/// Таблица элементов, только для чтения

/// </summary>

public ReadOnlyCollection<IdentitifierInfo> Items { get { return \_items.AsReadOnly(); } }

/// <summary>

/// Общее количество операций

/// </summary>

private int \_totalOperationCount;

public int TotalOperationCount

{

get { return \_totalOperationCount; }

set

{

\_totalOperationCount = value;

OnPropertyChanged(nameof(TotalOperationCount));

}

}

/// <summary>

/// Среднее количество операций

/// </summary>

private int \_middleOperationCount;

public int MiddleOperationCount

{

get { return \_middleOperationCount; }

set

{

\_middleOperationCount = value;

OnPropertyChanged(nameof(MiddleOperationCount));

}

}

/// <summary>

/// Время затраченное на выполнение одной из операций

/// </summary>

private string \_elapseTime;

public string ElapsedTime

{

get { return \_elapseTime; }

set

{

\_elapseTime = value;

OnPropertyChanged(nameof(ElapsedTime));

}

}

#endregion

#region Конструктор

/// <summary>

/// Конструктор по умолчанию

/// </summary>

public SimpleIndentityTable()

{

\_items = new List<IdentitifierInfo>();

timer = new Stopwatch();

}

#endregion

#region Реализация интерфейса

/// <summary>

/// Добавление элемента в таблицу

/// </summary>

/// <param name="value"></param>

public void AddItem(IdentitifierInfo value)

{

if (value == null)

{ throw new ArgumentNullException(nameof(value)); }

if (string.IsNullOrEmpty(value.ItemName))

{ throw new ArgumentNullException(nameof(value.ItemName)); }

if (value.ItemName.Length > MAX\_IDENTIFY\_LENGTH)

{ throw new ArgumentNullException(@"Длина ключа превышает лимит в

{MAX\_IDENTIFY\_LENGTH}"); }

foreach (IdentitifierInfo item in \_items)

{

TotalOperationCount++;

if (item.ItemName == value.ItemName)

{ throw new Exception("Элемент с таким ключом присутствует" +

value.ItemName); }

}

\_items.Add(value);

}

/// <summary>

/// Поиск элемента в таблице по ключу

/// </summary>

/// <param name="key">ключ</param>

/// <returns>Найденный объект или null</returns>

public IdentitifierInfo FindItem(string key)

{

if (string.IsNullOrEmpty(key))

{ throw new ArgumentNullException(nameof(key)); }

if (\_items == null)

{ throw new ArgumentNullException(nameof(\_items)); }

foreach (IdentitifierInfo item in \_items)

{

TotalOperationCount++;

if (item.ItemName == key)

{ return item; }

}

return null;

}

/// <summary>

/// Удаление элемента по ключу

/// </summary>

/// <param name="key"></param>

/// <returns></returns>

public bool RemoveItem(string key)

{

if (string.IsNullOrEmpty(key))

{ throw new ArgumentNullException(nameof(key)); }

if (\_items == null)

{ throw new ArgumentNullException(nameof(\_items)); }

foreach (IdentitifierInfo item in \_items)

{

TotalOperationCount++;

if (item.ItemName == key)

{ return \_items.Remove(item); }

}

return false;

}

/// <summary>

/// Очистка всей таблицы

/// </summary>

public void ClearItems()

{

if (\_items != null)

{

\_items.Clear();

}

}

#endregion

}

}