**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

по дисциплине: «Объектно-ориентированное проектирование и программирование»

на тему: **«**Коллекции. Обобщенные типы. Итераторы**»**

Выполнил: студент гр. ИТИ-21

Бахонько А. М.

Принял: преподаватель

Башаримов Ю. С.

Гомель 2024

**Цель работы**: изучить работу коллекций; узнать, как использовать итератор; выяснить методы и свойства внутри интерфейса *ICollection;* научится пользоваться обобщенными типами.

**Задание:**

• Разработать коллекцию**.**

• Обязательное наличие итератора.

• Необходимо реализовать интерфейс *ICollection*.

• Коллекция обязательно должна хранить любые типы данных (классы и структуры).

• Все коллекции должны быть реализованы без использования стандартных коллекций.

|  |  |
| --- | --- |
| № | Условие |
| 2 | Односвязный список. |

**Ход работы**

На рисунке 1 представлено окно с меню приложения.

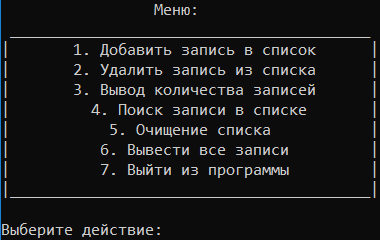


Рисунок 1 – Меню в консоли

Выберем действие 1, где добавим в список запись. Запись можно добавить либо в начало списка, либо в конец. Результат представлен на рисунке 2.

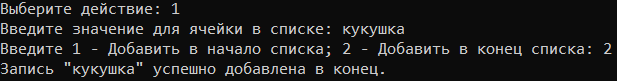


Рисунок 2 – Результат добавления записи

Выберем действие 2, где удалим из списка запись. Результат представлен на рисунке 3.

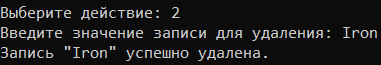


Рисунок 3 – Результат удаления записи

Теперь выберем действие 3, где выведем количество записей внутри списка. Результат представлен на рисунке 4.



Рисунок 4 – Результат вывода количества записей

Выберем действие 4, где найдем из списка выбранную запись. Результат представлен на рисунках 5.



Рисунок 5 – Результат поиска записи в списке

Выберем действие 6, где выведем все записи внутри списка от первой записи к последней. Результат представлен на рисунках 6.

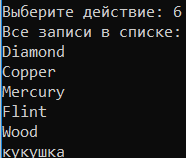


Рисунок 6 – Вывод всех записей

Выберем действие 5, где очистим список от записей. Результат представлен на рисунках 7.



Рисунок 7 – Вывод всех записей из очищенного списка

На рисунке 8 представлены результаты *unit*-тестов.

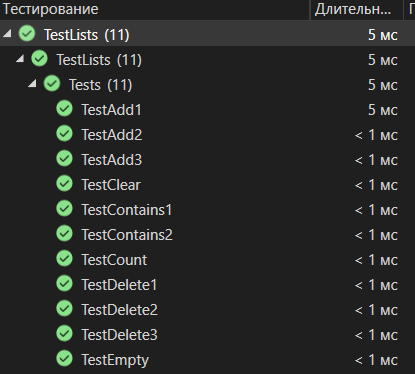


Рисунок 8 – Выполненные модульные тесты

В приложении А представлен код программы.

На рисунке 9 представлен репозиторий проекта.

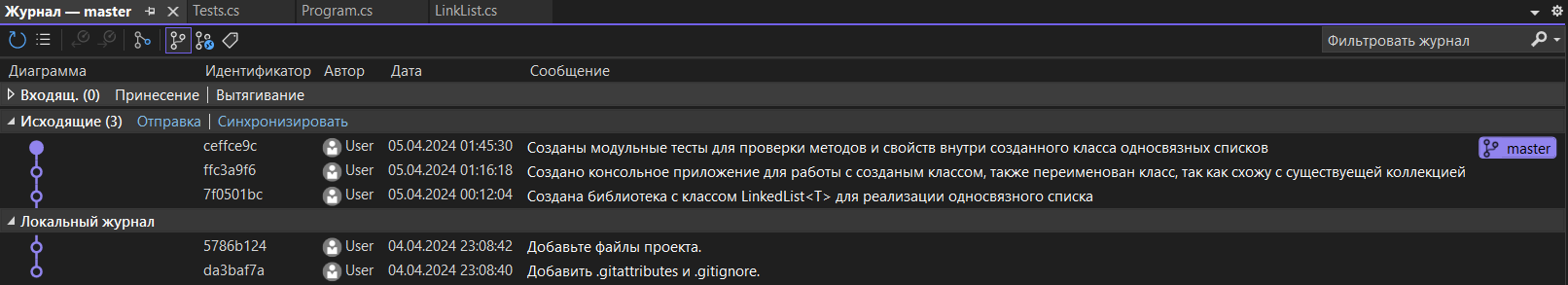


Рисунок 9 – Репозиторий проекта

**Вывод:** в результате выполнения лабораторной работы получили новые (и закрепили уже имеющиеся) знания работы со средой разработки *Visual Studio*, получили навыки создания коллекции для обобщенных типов данных из интерфейса *ICollection* и реализовали итератор для нашей коллекции данных.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Текст программы**

**LinkList.cs**

using System.Collections;

namespace LinkedListLibrary

{

public class LinkList<T> : ICollection<T>

{

private class Node<T>

{

public Node(T data)

{

Data = data;

}

public T Data { get; set; }

public Node<T>? Next { get; set; }

}

private Node<T>? head; // головной/первый элемент

private Node<T>? tail; // последний/хвостовой элемент

private int count; // количество элементов в списке

public int Count => count;

public bool IsReadOnly => false;

public bool IsEmpty => count == 0;

public void Add(T item)

{

Node<T> node = new Node<T>(item);

if (IsEmpty)

head = node;

else

tail!.Next = node;

tail = node;

count++;

}

// добвление в начало

public void AppendFirst(T data)

{

Node<T> node = new Node<T>(data);

node.Next = head;

head = node;

if (IsEmpty)

tail = head;

count++;

}

public void Clear()

{

head = null;

tail = null;

count = 0;

}

public bool Contains(T item)

{

Node<T>? current = head;

while (current != null && current.Data != null)

{

if (current.Data.Equals(item)) return true;

current = current.Next;

}

return false;

}

public void CopyTo(T[] array, int arrayIndex)

{

if (!IsEmpty)

{

Node<T>? node = head;

array[arrayIndex++] = node!.Data;

node = node.Next;

while (node != null && node.Data != null)

{

array[arrayIndex++] = node.Data;

}

}

}

public IEnumerator<T> GetEnumerator()

{

Node<T>? current = head;

while (current != null)

{

yield return current.Data;

current = current.Next;

}

}

public bool Remove(T item)

{

Node<T>? current = head;

Node<T>? previous = null;

while (current != null && current.Data != null)

{

if (current.Data.Equals(item))

{

// Если узел в середине или в конце

if (previous != null)

{

// убираем узел current, теперь previous ссылается не на current, а на current.Next

previous.Next = current.Next;

// Если current.Next не установлен, значит узел последний,

// изменяем переменную tail

if (current.Next == null)

tail = previous;

}

else

{

// если удаляется первый элемент

// переустанавливаем значение head

head = head?.Next;

// если после удаления список пуст, сбрасываем tail

if (head == null)

tail = null;

}

count--;

return true;

}

previous = current;

current = current.Next;

}

return false;

}

IEnumerator IEnumerable.GetEnumerator()

{

return ((IEnumerable<T>)this).GetEnumerator();

}

}

}

**Program.cs**

using LinkedListLibrary;

class Program

{

static LinkList<string> linkedList = new LinkList<string>();

static void Main(string[] args)

{

// Добавляем записи

linkedList.Add("Diamond");

linkedList.Add("Iron");

linkedList.Add("Copper");

linkedList.Add("Mercury");

linkedList.Add("Flint");

linkedList.Add("Wood");

while (true)

{

Console.WriteLine("\n Меню:");

Console.WriteLine(" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

Console.WriteLine("| 1. Добавить запись в список |");

Console.WriteLine("| 2. Удалить запись из списка |");

Console.WriteLine("| 3. Вывод количества записей |");

Console.WriteLine("| 4. Поиск записи в списке |");

Console.WriteLine("| 5. Очищение списка |");

Console.WriteLine("| 6. Вывести все записи |");

Console.WriteLine("| 7. Выйти из программы |");

Console.WriteLine("|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n");

Console.Write("Выберите действие: ");

string choice = Console.ReadLine();

switch (choice)

{

case "1":

AddNode();

break;

case "2":

RemoveNode();

break;

case "3":

Console.WriteLine($"Количество записей в списке равно: {linkedList.Count}.");

break;

case "4":

SearchNode();

break;

case "5":

linkedList.Clear();

break;

case "6":

PrintAllNodes();

break;

case "7":

Environment.Exit(0);

break;

default:

Console.WriteLine("Неправильный выбор. Пожалуйста, введите число от 1 до 7.");

break;

}

Console.WriteLine();

}

}

static void AddNode()

{

Console.Write("Введите значение для ячейки в списке: ");

string value = Console.ReadLine();

if (string.IsNullOrWhiteSpace(value))

{

Console.WriteLine("Значение записи не может быть пустым.");

}

else

{

bool isAdd = false;

while (!isAdd)

{

Console.Write("Введите 1 - Добавить в начало списка; 2 - Добавить в конец списка: ");

string number = Console.ReadLine();

switch (number)

{

case "1":

linkedList.AppendFirst(value);

Console.WriteLine($"Запись \"{value}\" успешно добавлена в начало.");

isAdd = true;

break;

case "2":

linkedList.Add(value);

Console.WriteLine($"Запись \"{value}\" успешно добавлена в конец.");

isAdd = true;

break;

default:

Console.WriteLine("Вводимое значение не входит в выбор действия.");

break;

}

}

}

}

static void RemoveNode()

{

Console.Write("Введите значение записи для удаления: ");

string value = Console.ReadLine();

if (!string.IsNullOrWhiteSpace(value))

{

if (linkedList.Remove(value))

Console.WriteLine($"Запись \"{value}\" успешно удалена.");

else

Console.WriteLine($"Запись \"{value}\" не найдена в списке.");

}

else

{

Console.WriteLine("Значение записи не может быть пустым.");

}

}

static void SearchNode()

{

Console.Write("Введите значение для поиска в списке: ");

string node = Console.ReadLine();

if (!string.IsNullOrWhiteSpace(node))

{

if (linkedList.Contains(node))

Console.WriteLine($"Запись {node} есть в списке.");

else

Console.WriteLine($"Записи {node} нет в списке");

}

else

{

Console.WriteLine("Значения узлов не могут быть пустыми.");

}

}

static void PrintAllNodes()

{

Console.WriteLine("Все записи в списке:");

foreach (var node in linkedList)

{

Console.WriteLine(node);

}

}

}

**Tests.cs**

using LinkedListLibrary;

namespace TestLists

{

[TestClass]

public class Tests

{

[TestMethod]

public void TestAdd1()

{

LinkList<string> result = new LinkList<string>();

result.Add("a");

LinkList<string> list = new LinkList<string>() { "a" };

Assert.AreEqual(list.Contains("a"), result.Contains("a"));

}

[TestMethod]

public void TestAdd2()

{

LinkList<string> result = new LinkList<string>();

result.Add("prev");

Assert.IsTrue(result.Contains("prev"));

}

[TestMethod]

public void TestAdd3()

{

LinkList<string> result = new LinkList<string>();

result.Add("prev");

result.Add("word");

result.Add("mist");

result.Add("more");

Assert.IsTrue(result.Contains("prev") && result.Contains("word") && result.Contains("mist") && result.Contains("more"));

}

[TestMethod]

public void TestDelete1()

{

LinkList<string> result = new LinkList<string>() { "prev" };

Assert.IsTrue(result.Remove("prev"));

}

[TestMethod]

public void TestDelete2()

{

LinkList<string> result = new LinkList<string>() { "prev", "mist", "link" };

Assert.IsTrue(result.Remove("prev") && result.Remove("mist") && result.Remove("link"));

}

[TestMethod]

public void TestDelete3()

{

LinkList<string> result = new LinkList<string>();

Assert.IsFalse(result.Remove("prev"));

}

[TestMethod]

public void TestClear()

{

LinkList<string> result = new LinkList<string>() { "prev", "mist", "link" };

result.Clear();

Assert.IsFalse(result.Contains("prev") && result.Contains("mist") && result.Contains("link"));

}

[TestMethod]

public void TestCount()

{

LinkList<string> result = new LinkList<string>() { "prev", "mist", "link" };

Assert.AreEqual(3, result.Count);

}

[TestMethod]

public void TestEmpty()

{

LinkList<string> result = new LinkList<string>() { "prev", "mist", "link" };

Assert.IsFalse(result.IsEmpty);

}

[TestMethod]

public void TestContains1()

{

LinkList<string> result = new LinkList<string>() { "prev", "mist", "link" };

Assert.IsTrue(result.Contains("prev") && result.Contains("mist") && result.Contains("link"));

}

[TestMethod]

public void TestContains2()

{

LinkList<string> result = new LinkList<string>();

Assert.IsFalse(result.Contains("prev") && result.Contains("mist") && result.Contains("link"));

}

}

}