

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Вычислительной техники

ОТЧЕТ
по лабораторной работе №1
по дисциплине «Программирование в среде dotNET»
Тема: «РЕАЛИЗАЦИЯ БАЗОВЫХ АЛГОРИТМОВ СРЕДСТВАМИ
ЯЗЫКА C#»

Студент гр. 6305

Белоусов Е.О.

Преподаватель

Пешехонов К.А.

Санкт-Петербург

2020

СОДЕРЖАНИЕ

ЦЕЛЬ РАБОТЫ И ЗАДАНИЕ	3
ХОД РАБОТЫ	4
Связный список	4
Бинарное дерево	5
Сортировка вставками.....	6
ВЫВОДЫ	7
ПРИЛОЖЕНИЕ	8
Связный список	8
Бинарное дерево	13
Сортировка вставками.....	18

ЦЕЛЬ РАБОТЫ И ЗАДАНИЕ

Цель работы: ознакомиться с базовыми средствами языка C#.

Постановка задания:

1. Реализовать связный список (без использования стандартных коллекций/LINQ, кроме IEnumerable) со следующими операциями: создание, удаление, добавление произвольных элементов, реверс списка.
2. Реализовать бинарное дерево (без использования стандартных деревьев) со следующими операциями: поиск элемента, удаление элемента.
3. Реализовать сортировку вставками (без использования метода OrderBy()).

ХОД РАБОТЫ

Связный список

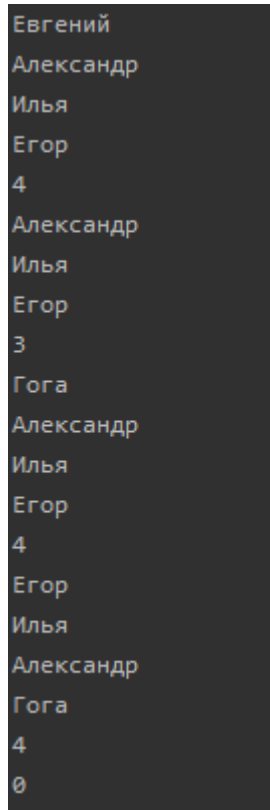
Связный список представляет набор связанных узлов, каждый из которых хранит собственно данные и ссылку на следующий узел.

Реализация представлена в приложении к отчёту.

Рассмотрим работу реализованной программы. Выполним операции в следующем порядке:

1. Добавляем элементы.
2. Выводим элементы.
3. Выводим количество элементов.
4. Удаляем элемент и выводим оставшиеся элементы.
5. Выводим количество элементов.
6. Добавляем элемент в начало и выводим элементы.
7. Выводим количество элементов.
8. Реверсируем список и выводим элементы.
9. Выводим количество элементов.
10. Очищаем список и выводим количество элементов.

Результат представлен на рисунке 1.



```
Евгений
Александр
Илья
Егор
4
Александр
Илья
Егор
3
Гога
Александр
Илья
Егор
4
Егор
Илья
Александр
Гога
4
0
```

Рисунок 1. Работа связного списка

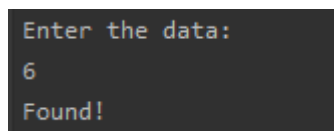
Бинарное дерево

Бинарное дерево – это динамическая структура данных, представляющая собой дерево, в котором каждая вершина имеет не более двух потомков. Таким образом, бинарное дерево состоит из элементов, каждый из которых содержит информационное поле и не более двух ссылок на различные бинарные поддеревья.

Реализация представлена в приложении к отчёту.

Рассмотрим работу реализованной программы:

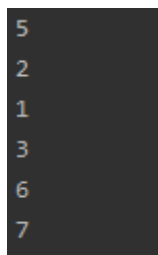
1. Введём в следующем порядке числа: 5, 2, 3, 1, 6, 7. Попробуем найти элемент «6» (рисунок 2).



```
Enter the data:
6
Found!
```

Рисунок 2. Бинарное дерево: поиск элемента

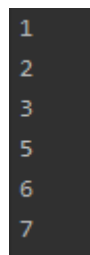
2. Выведем элементы с помощью прямого (pre-order) обхода (рисунок 3).



```
5
2
1
3
6
7
```

Рисунок 3. Бинарное дерево: прямой (pre-order) обход

3. Выведем элементы с помощью симметричного (in-order) обхода (рисунок 4).



```
1
2
3
5
6
7
```

Рисунок 4. Бинарное дерево: симметричный (in-order) обход

4. Выведем элементы с помощью обратного (post-order) обхода (рисунок 5).

```
1
3
2
7
6
5
```

Рисунок 5. Бинарное дерево: обратный (post-order) обход

5. Удалим элемент «7» и выведем оставшиеся элементы с помощью симметричного обхода (рисунок 6).

```
Enter the data:
7

1
2
3
5
6
```

Рисунок 6. Бинарное дерево: удаление элемента и вывод оставшихся симметричным обходом

Сортировка вставками

Сортировка вставками — это алгоритм сортировки массивов, в котором на каждой итерации первый элемент неупорядоченной последовательности помещается в подходящее место среди последовательности ранее упорядоченных элементов.

Реализация представлена в приложении к отчёту.

Рассмотрим работу реализованной программы. Введём в следующем порядке числа: 4, 11, 8, 1, 0, 2, 3, 9, 7. Результат представлен на рисунке 7.

```
Please, enter the values: 4 11 8 1 0 2 3 9 7
Sorted by insertions: 0 1 2 3 4 7 8 9 11
```

Рисунок 7. Сортировка вставками

ВЫВОДЫ

В результате выполнения данной лабораторной работы были изучены базовые средства языка C#.

ПРИЛОЖЕНИЕ

Связный список

```
/**
 * Дисциплина: Основы разработки корпоративных приложений на платформе .NET
 * Тема: Тестовое задание #1 - Связный список (без стандартных коллекций и LINQ).
 * Разработал: Белоусов Евгений
 * Группа: 6305
 */
using System;
using System.Collections;
using System.Collections.Generic;

namespace TestLinkedList
{
    /// <summary>
    /// Класс элемента связного списка.
    /// </summary>
    /// <remarks>
    /// Каждый объект класса содержит некоторую информацию
    /// и ссылку на следующий элемент списка.
    /// </remarks>
    public class Node<T>
    {
        public T Data { get; }
        public Node<T> Next { get; set; }

        public Node(T data)
        {
            Data = data;
        }
    }

    /// <summary>
    /// Класс односвязного списка.
    /// </summary>
    /// <remarks>
    /// Класс реализует интерфейс IEnumerable.
    /// В каждом объекте сохраняются ссылки на начало и конец списка,
    /// а также учитывается количество содержащихся элементов.
    /// Реализуются типовые действия со связным списком:
    /// - добавление произвольных данных (в начало и в конец);
    /// - удаление произвольных данных (при первом их вхождении);
    /// - реверс списка.
    /// </remarks>
    public class LinkedList<T> : IEnumerable<T>
    {
        private Node<T> _head;
        private Node<T> _tail;
```



```

private int _count;

/// <summary>
/// Добавить данные в конец связанного списка.
/// </summary>
/// <param name="data"> Произвольные данные. </param>
public void Add(T data)
{
    if (data == null)
        throw new ArgumentNullException(nameof(data));

    var node = new Node<T>(data);
    if (_head == null)
        _head = node;
    else
        _tail.Next = node;

    _tail = node;
    _count++;
}

/// <summary>
/// Добавить данные в начало связанного списка.
/// </summary>
/// <param name="data"> Произвольные данные. </param>
public void AddFirst(T data)
{
    var node = new Node<T>(data);
    node.Next = _head;
    _head = node;
    if (IsEmpty)
        _tail = _head;

    _count++;
}

/// <summary>
/// Удалить данные из связанного списка.
/// </summary>
/// <remarks>
/// Удаляется первое вхождение данных.
/// </remarks>
/// <param name="data"> Произвольные данные. </param>
public bool Remove(T data)
{
    Node<T> current = _head;
    Node<T> previous = null;
    while (current != null)
    {
        if (data != null && current.Data.Equals(data))

```

```

    {
        if (previous != null)
        {
            previous.Next = current.Next;
            if (current.Next == null)
                _tail = previous;
        }
        else
        {
            _head = _head.Next;
            if (_head == null)
                _tail = null;
        }

        _count--;
        return true;
    }

    previous = current;
    current = current.Next;
}

return false;
}

/// <summary>
/// Реверсировать список.
/// </summary>
public void Reverse()
{
    Node<T> current = _head;
    Node<T> previous = null;
    Node<T> next;

    while (current != null)
    {
        next = current.Next;
        if (previous != null)
        {
            current.Next = previous;
        }
        else
        {
            current.Next = _tail.Next;
            _tail = current;
        }

        previous = current;
        current = next;
    }
}

```

```

        _head = previous;
    }

    /// <summary>
    /// Очистить список полностью.
    /// </summary>
    public void Clear()
    {
        _head = null;
        _tail = null;
        _count = 0;
    }

    /// <summary>
    /// Свойство, предназначенное для проверки наличия в списке элементов.
    /// </summary>
    public bool IsEmpty => _count == 0;

    /// <summary>
    /// Свойство, предназначенное для получения количества элементов,
    /// содержащихся в списке.
    /// </summary>
    public int Count => _count;

    /// <summary>
    /// Вернуть перечислитель, выполняющий перебор всех элементов в связанном
    списке.
    /// </summary>
    /// <remarks>
    /// Реализация интерфейса IEnumerable.
    /// </remarks>
    /// <returns> Возвращает перечислитель. </returns>
    public IEnumerator<T> GetEnumerator()
    {
        var current = _head;
        while (current != null)
        {
            yield return current.Data;
            current = current.Next;
        }
    }

    /// <summary>
    /// Вернуть перечислитель, осуществляющий итерационный переход по связанному
    списку.
    /// </summary>
    /// <remarks>
    /// Реализация интерфейса IEnumerable.
    /// </remarks>
    /// <returns> Объект IEnumerator. </returns>

```

```

IEnumerator IEnumerable.GetEnumerator()
{
    return (this as IEnumerable).GetEnumerator();
}
}

public static class Program
{
    /// <summary>
    /// Точка входа программы.
    /// </summary>
    /// <param name="args"> Список аргументов командной строки.</param>
    public static void Main(string[] args)
    {
        var linkedList = new LinkedList<string>();
        // добавляем элементы
        linkedList.Add("Евгений");
        linkedList.Add("Александр");
        linkedList.Add("Илья");
        linkedList.Add("Егор");
        // выводим элементы
        foreach(var item in linkedList)
            Console.WriteLine(item);
        // выводим количество элементов
        Console.WriteLine(linkedList.Count);
        // удаляем элемент и выводим оставшиеся элементы
        linkedList.Remove("Евгений");
        foreach (var item in linkedList)
            Console.WriteLine(item);
        // выводим количество элементов
        Console.WriteLine(linkedList.Count);
        // добавляем элемент в начало и выводим элементы
        linkedList.AddFirst("Гога");
        foreach(var item in linkedList)
            Console.WriteLine(item);
        // выводим количество элементов
        Console.WriteLine(linkedList.Count);
        // реверсируем список и выводим элементы
        linkedList.Reverse();
        foreach(var item in linkedList)
            Console.WriteLine(item);
        // выводим количество элементов
        Console.WriteLine(linkedList.Count);
        // очищаем список и выводим количество элементов
        linkedList.Clear();
        Console.WriteLine(linkedList.Count);
    }
}
}

```

Бинарное дерево

```
/**
 * Дисциплина: Основы разработки корпоративных приложений на платформе .NET
 * Тема: Тестовое задание #2 - Бинарное дерево (без стандартных деревьев)
 * Разработал: Белоусов Евгений
 * Группа: 6305
 */
using System;

namespace TestBinaryTree
{
    public class Node
    {
        public int Value;
        public Node Left;
        public Node Right;
    }

    public class BinaryTree
    {
        private Node _root;

        public BinaryTree()
        {
            _root = null;
        }

        public void InsertNode(int key)
        {
            if (_root != null)
                InsertNode(key, _root);
            else
            {
                _root = new Node
                {
                    Value = key,
                    Left = null,
                    Right = null
                };
            }
        }

        private void InsertNode(int key, Node leaf)
        {
            if (leaf == null)
                throw new ArgumentNullException(nameof(leaf));

            while (true)
            {
                if (key < leaf.Value)
                {

```

```

        if (leaf.Left != null)
        {
            leaf = leaf.Left;
            continue;
        }

        leaf.Left = new Node
        {
            Value = key,
            Left = null,
            Right = null
        };
    }
    else if (key >= leaf.Value)
    {
        if (leaf.Right != null)
        {
            leaf = leaf.Right;
            continue;
        }

        leaf.Right = new Node
        {
            Value = key,
            Right = null,
            Left = null
        };
    }

    break;
}
}

public Node SearchNode(int key)
{
    return SearchNode(key, _root);
}

private static Node SearchNode(int key, Node leaf)
{
    while (true)
    {
        if (leaf != null)
        {
            if (key == leaf.Value)
                return leaf;

            leaf = key < leaf.Value ? leaf.Left : leaf.Right;
        }
        else

```

```

        {
            return null;
        }
    }
}

public void RemoveNode(int key)
{
    RemoveNode(_root, SearchNode(key, _root));
}

private static Node RemoveNode(Node root, Node removableNode)
{
    if (root == null)
        return null;

    if (removableNode.Value < root.Value)
        root.Left = RemoveNode(root.Left, removableNode);

    if (removableNode.Value > root.Value)
        root.Right = RemoveNode(root.Right, removableNode);

    if (removableNode.Value != root.Value)
        return root;

    switch (root.Left)
    {
        case null when root.Right == null:
        {
            return null;
        }
        case null:
        {
            root = root.Right;
            break;
        }
        default:
        {
            if (root.Right == null)
            {
                root = root.Left;
            }
            else
            {
                var minimalNode = GetMinimalNode(root.Right);
                root.Value = minimalNode.Value;
                root.Right = RemoveNode(root.Right, minimalNode);
            }

            break;
        }
    }
}

```

```

        }
    }

    return root;
}

private static Node GetMinimalNode(Node currentNode)
{
    while (currentNode?.Left != null)
        currentNode = currentNode.Left;

    return currentNode;
}

public void PreOrderTravers()
{
    PreOrderTravers(_root);
    Console.WriteLine("");
}

private static void PreOrderTravers(Node leaf)
{
    while (true)
    {
        if (leaf == null)
            return;
        Console.WriteLine("{0}", leaf.Value);
        PreOrderTravers(leaf.Left);
        leaf = leaf.Right;
    }
}

public void InOrderTravers()
{
    InOrderTravers(_root);
    Console.WriteLine("");
}

private static void InOrderTravers(Node leaf)
{
    while (true)
    {
        if (leaf != null)
        {
            InOrderTravers(leaf.Left);
            Console.WriteLine("{0}", leaf.Value);
            leaf = leaf.Right;
            continue;
        }
    }
}

```



```

        break;
    }
}

public void PostOrderTravers()
{
    PostOrderTravers(_root);
    Console.WriteLine("");
}

private static void PostOrderTravers(Node leaf)
{
    if (leaf == null)
        return;
    PostOrderTravers(leaf.Left);
    PostOrderTravers(leaf.Right);
    Console.WriteLine("{0}", leaf.Value);
}
}

public class Program
{
    public static void Main(string[] args)
    {
        var tree = new BinaryTree();

        while (true)
        {
            Console.WriteLine("What you going to do?");
            Console.WriteLine("Enter the number:");
            Console.WriteLine("1 - Insert;");
            Console.WriteLine("2 - Remove;");
            Console.WriteLine("3 - Search;");
            Console.WriteLine("4 - Pre-order travers;");
            Console.WriteLine("5 - In order travers;");
            Console.WriteLine("6 - Post-order travers;");
            Console.WriteLine("0 - Exit.");

            var data = "";
            switch (Convert.ToInt32(Console.ReadLine()))
            {
                case 1:
                    Console.WriteLine("Enter the data:");
                    data = Console.ReadLine();
                    tree.InsertNode(Convert.ToInt32(data));
                    break;
                case 2:
                    Console.WriteLine("Enter the data:");
                    data = Console.ReadLine();
                    tree.RemoveNode(Convert.ToInt32(data));

```



```

    {
        for (var index = 1; index < array.Length; index++)
        {
            var key = array[index];
            var indexOfSorted = index;
            while (indexOfSorted > 0 && array[indexOfSorted - 1] > key)
            {
                Swap(ref array[indexOfSorted - 1], ref array[indexOfSorted]);
                indexOfSorted--;
            }

            array[indexOfSorted] = key;
        }
        return array;
    }

    /// <summary>
    /// Метод меняет значения двух целочисленных переменных между собой.
    /// </summary>
    /// <remarks>
    /// Переменные передаются в метод по ссылке.
    /// </remarks>
    /// <param name="valueA"> Первая целочисленная переменная.</param>
    /// <param name="valueB"> Вторая целочисленная переменная.</param>
    private static void Swap(ref int valueA, ref int valueB)
    {
        var temp = valueA;
        valueA = valueB;
        valueB = temp;
    }

    /// <summary>
    /// Точка входа программы.
    /// </summary>
    /// <param name="args"> Список аргументов командной строки.</param>
    public static void Main(string[] args)
    {
        Console.WriteLine("Please, enter the values: ");
        var values = Console.ReadLine()?.Split(new[] { " ", "," },
StringSplitOptions.RemoveEmptyEntries);
        var array = new int[values.Length];
        for (var index = 0; index < values.Length; index++)
            array[index] = Convert.ToInt32(values[index]);

        Console.WriteLine("Sorted by insertions: {0}", string.Join(" ",
InsertionSort(array)));
    }
}

```