**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра вычислительной техники**

**ОТЧЁТ**

**по лабораторной работе №1**

**по дисциплине «Программирование в среде .NET»**

**Тема: “Реализация базовых алгоритмов на языке программирования C#”**

|  |  |  |
| --- | --- | --- |
| Студент гр. 6305 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Буракаев Д. А. |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Пешехонов К. А. |

Санкт-Петербург

2020

**Содержание**

[**Цель работы** 3](#_Toc38617272)

[**Задание** 3](#_Toc38617273)

[**Ход работы** 4](#_Toc38617274)

[**Связанный список** 4](#_Toc38617275)

[**Бинарное дерево** 5](#_Toc38617276)

[**Сортировка вставками** 8](#_Toc38617277)

[**Вывод** 9](#_Toc38617278)

[**Приложение А. Связанный список** 10](#_Toc38617279)

[**Приложение Б. Бинарное дерево** 15](#_Toc38617280)

[**Приложение В. Сортировка вставками** 20](#_Toc38617281)

# **Цель работы**

Целью данной лабораторной работы является ознакомление с базовыми средствами разработки консольных приложений на языке C# в среде программирования *Visual Studio*.

# **Задание**

Рекомендуется придерживаться следующего порядка работы.

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.

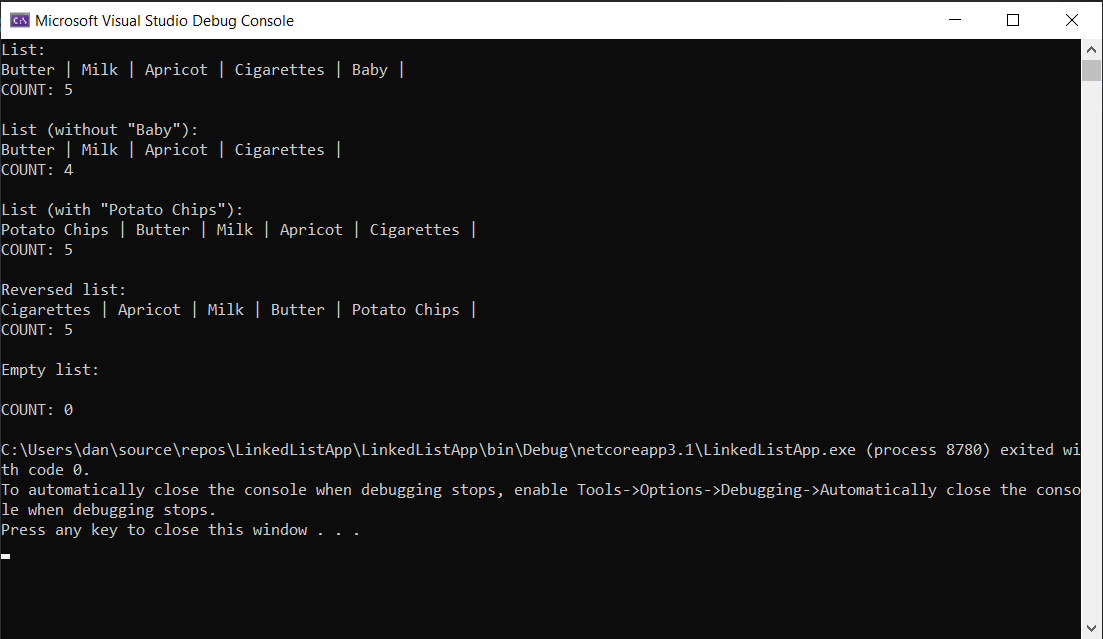
3. Реализовать сортировку вставками (без использования метода OrderBy()).

# **Ход работы**

## **Связанный список**

В ходе выполнения лабораторной работы была написана программа на языке *C#* (листинг программы представлен в **Приложении А**). Полученная программа будет производить создание и обработку односвязного списка.

Рассмотрим процесс исполнения разработанного приложения:



*Рис 1. Манипулирование односвязным списком*

По ходу выполнения программы в экземпляр класса списка вносятся следующие строковые значения: «*Butter*», «*Milk*», «*Apricot*», «*Cigarettes*», «*Baby*».

После этого производится удаление из списка последнего объекта «*Baby*» и выводится список уже без этого объекта.

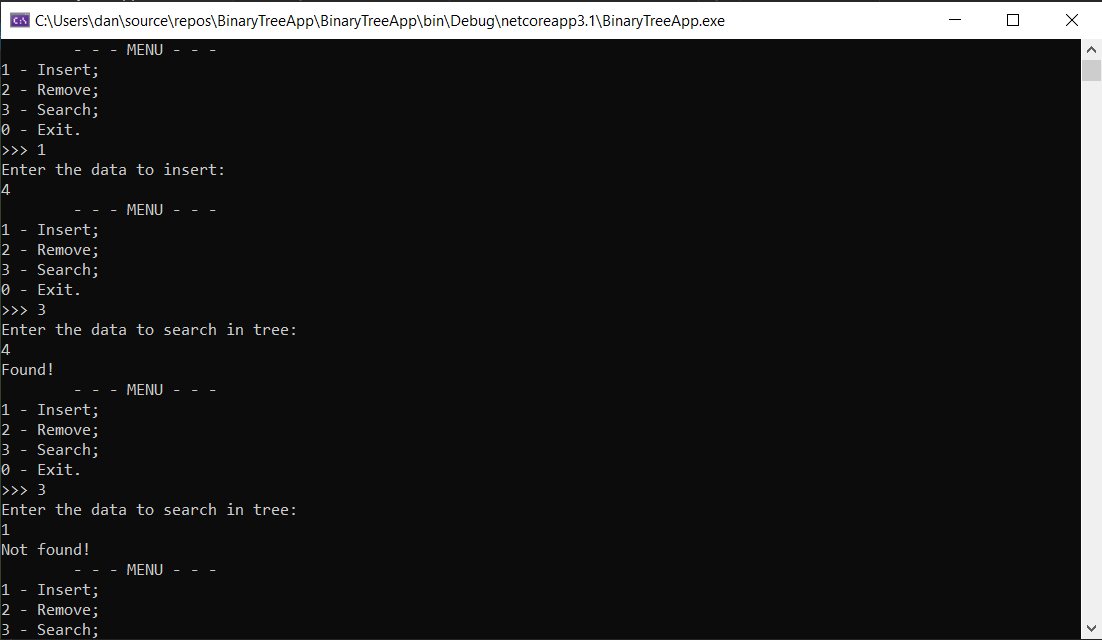
Затем в список добавляется значение «*Potato Chips*» и он вновь выводится на консоль, чтобы продемонстрировать корректность исполнения метода добавления нового элемента односвязного списка.

Под конец своей работы программа реверсирует полученный список и демонстрирует его на консоли, чтобы потом его полностью очистить.

## **Бинарное дерево**

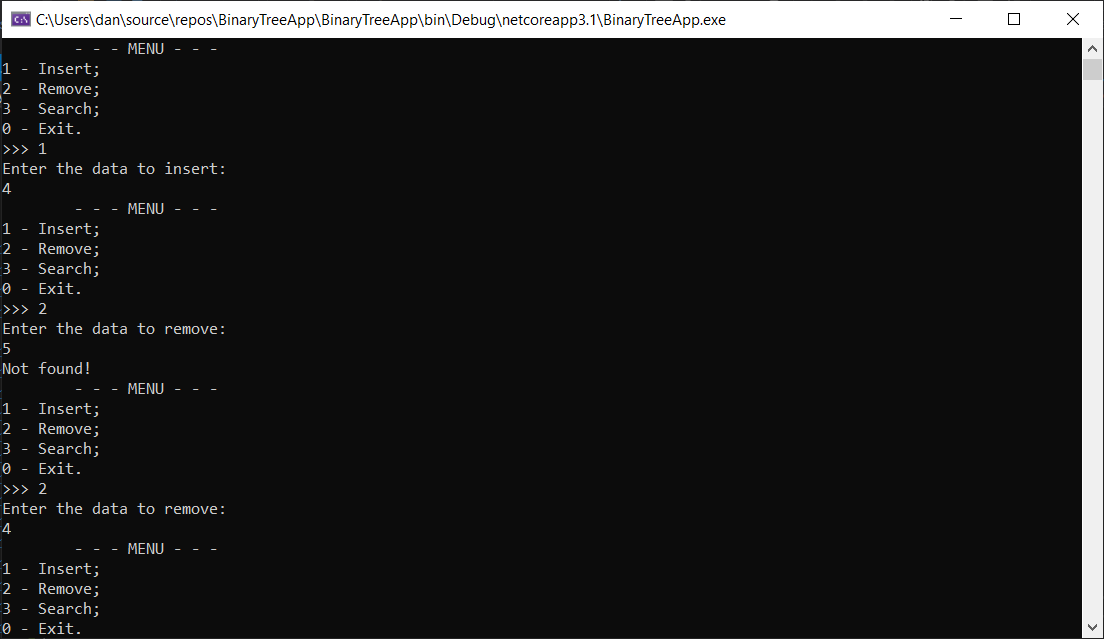
В ходе выполнения лабораторной работы была написана программа на языке *C#* (листинг программы представлен в **Приложении Б**). Полученная программа создавать из класса объект-экземпляр бинарного дерева поиска, наполнять его, удалять элементы и производить поиск.

Рассмотрим процесс исполнения разработанного приложения:



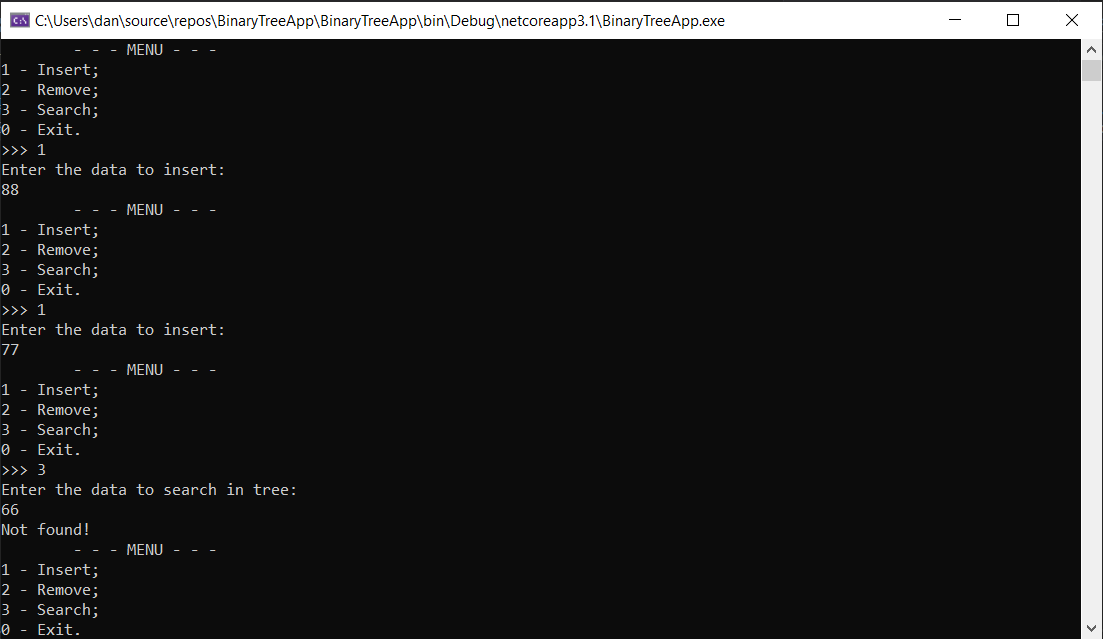
*Рис 2. Добавление элемента*

Как видно на скриншоте, после добавления числа 4 в дерево, существует возможность его в этом дереве найти.



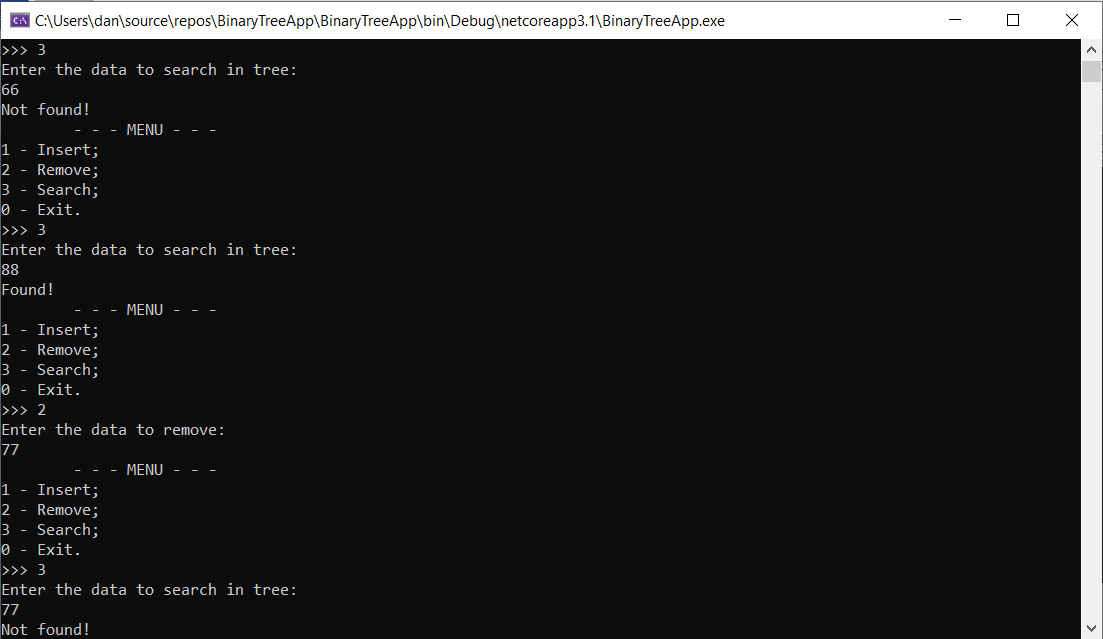
*Рис 3. Удаление элемента*

Как видно на скриншоте, после добавления 4 в бинарное дерево была произведена попытка удаления из него цифры 5. Программа выдала на консоль отрицательный ответ, указывая на то, что такой цифры нет в дереве. После было успешно проведено удаление 4 из дерева.



*Рис 4. Поиск в существующем дереве*

На скриншоте видно, как после внесения чисел 77 и 88 была произведена попытка найти в бинарном дереве число 66. Программа выдала соответствующе сообщение, сигнализирующее об отсутствии искомого числа в дереве.



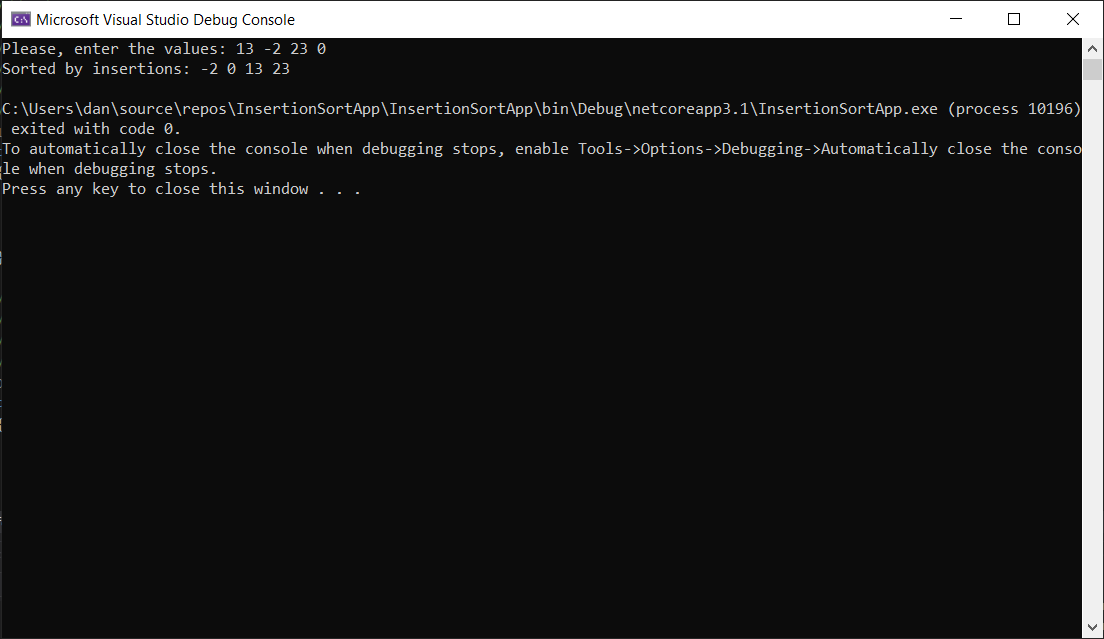
*Рис 5. Еще одна попытка поиска в существующем дереве*

После была попытка найти число 88, на что программа ответила утвердительным сообщением. После из бинарного дерева было удалено число 77 (теперь оно включает в себя только число 88), а затем попытка найти это число оказалась неудачна. Программа работает верно.

## **Сортировка вставками**

В ходе выполнения лабораторной работы была написана программа на языке C# (листинг программы представлен в **Приложении В**). Полученная программа реализует алгоритм сортировки массивов, при котором на каждой итерации первый элемент неупорядоченной последовательности помещается в подходящее место среди последовательности ранее упорядоченных элементов.

Рассмотрим процесс исполнения разработанного приложения:



*Рис 6. Сортировка вставками*

Как видно на скриншоте, программа успешно произвела сортировку внесенного через терминал массива целых чисел.

# **Вывод**

В ходе выполнения лабораторной работы было произведено ознакомление с базовыми средствами разработки консольных приложений на языке C# в среде программирования *Visual Studio*. Полученные программы были реализованы в соответствии с заданием и проверены.

# **Приложение А. Связанный список**

using System;

using System.Collections;

using System.Collections.Generic;

namespace LinkedListApp

{

/// <summary>

/// Object of page of list

/// </summary>

/// <remarks>

/// Each page has r-link to the next one

/// </remarks>

public class Node<T>

{

public T Data { get; }

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

public Node(T data)

{

Data = data;

}

}

/// <summary>

/// Singly linked list object

/// </summary>

/// <remarks>

/// Implements class IEnumerable.

/// Private variables:

/// \* fst - First page of the list (head)

/// \* lst - Last page of the list

/// \* n - Contains amount of pages in list

/// Methods:

/// \* Add(page) - Add as last page of the list

/// \* AddFirst(page) - Add as first page of the list

/// \* Remove(page) - Remove page

/// \* Reverse() - Reverse list

/// \* Clear() - Clear list

/// \* IsEmpty() - Return "true" if list has 0 pages

/// \* Count() - Return n variable

/// </remarks>

public class LinkedList<T> : IEnumerable<T>

{

private Node<T> fst;

private Node<T> lst;

private int n;

/// <summary>

/// Add as last page of the list

/// </summary>

/// <param name="data"> Page to add </param>

public void Add(T data)

{

if (data == null)

throw new ArgumentNullException(nameof(data));

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

if (fst == null)

fst = node;

else

lst.Next = node;

lst = node;

n++;

}

/// <summary>

/// Add as first page of the list

/// </summary>

/// <param name="data"> Page to add </param>

public void AddFirst(T data)

{

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

node.Next = fst;

fst = node;

if (IsEmpty)

lst = fst;

n++;

}

/// <summary>

/// Remove page

/// </summary>

/// <remarks>

/// Removes first detected page

/// </remarks>

/// <param name="data"> Page to remove from list </param>

public bool Remove(T data)

{

Node<T> current = fst;

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)

lst = previous;

}

else

{

fst = fst.Next;

if (fst == null)

lst = null;

}

n--;

return true;

}

previous = current;

current = current.Next;

}

return false;

}

/// <summary>

/// Reverse this list

/// </summary>

public void Reverse()

{

Node<T> current = fst;

Node<T> previous = null;

Node<T> next;

while (current != null)

{

next = current.Next;

if (previous != null)

{

current.Next = previous;

}

else

{

current.Next = lst.Next;

lst = current;

}

previous = current;

current = next;

}

fst = previous;

}

/// <summary>

/// Clear this list

/// </summary>

public void Clear()

{

fst = null;

lst = null;

n = 0;

}

/// <summary>

/// To check if list is empty

/// </summary>

public bool IsEmpty => n == 0;

/// <summary>

/// To check number of pages in list

/// </summary>

public int Count => n;

/// <summary>

/// Return IEnumerator which enumerates pages in list

/// </summary>

public IEnumerator<T> GetEnumerator()

{

var current = fst;

while (current != null)

{

yield return current.Data;

current = current.Next;

}

}

/// <summary>

/// Return IEnumerator

/// </summary>

IEnumerator IEnumerable.GetEnumerator()

{

return (this as IEnumerable).GetEnumerator();

}

}

public static class Program

{

/// <summary>

/// Program's Main foo

/// </summary>

/// <param name="args"> Terminal args </param>

public static void Main(string[] args)

{

var listToBuy = new LinkedList<string>();

// Fill the list

listToBuy.Add("Butter");

listToBuy.Add("Milk");

listToBuy.Add("Apricot");

listToBuy.Add("Cigarettes");

listToBuy.Add("Baby");

// Display list content

Console.WriteLine("List: ");

foreach (var item in listToBuy)

Console.Write(item + " | ");

Console.WriteLine("\nCOUNT: " + listToBuy.Count);

// Remove something and display

listToBuy.Remove("Baby");

Console.WriteLine("\nList (without \"Baby\"): ");

foreach (var item in listToBuy)

Console.Write(item + " | ");

Console.WriteLine("\nCOUNT: " + listToBuy.Count);

// Add to head of list

listToBuy.AddFirst("Potato Chips");

Console.WriteLine("\nList (with \"Potato Chips\"): ");

foreach (var item in listToBuy)

Console.Write(item + " | ");

Console.WriteLine("\nCOUNT: " + listToBuy.Count);

// Reverse and display

listToBuy.Reverse();

Console.WriteLine("\nReversed list: ");

foreach (var item in listToBuy)

Console.Write(item + " | ");

Console.WriteLine("\nCOUNT: " + listToBuy.Count);

// Clear and display

listToBuy.Clear();

Console.WriteLine("\nEmpty list: ");

foreach (var item in listToBuy)

Console.Write(item + " | ");

Console.WriteLine("\nCOUNT: " + listToBuy.Count);

}

}

}

# **Приложение Б. Бинарное дерево**

using System;

namespace BinaryTreeApp

{

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 == null)

{

Console.WriteLine("Not found!");

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 class Program

{

public static void Main(string[] args)

{

var tree = new BinaryTree();

while (true)

{

Console.WriteLine("\t- - - MENU - - -");

Console.WriteLine("1 - Insert;");

Console.WriteLine("2 - Remove;");

Console.WriteLine("3 - Search;");

Console.WriteLine("0 - Exit.");

Console.Write(">>> ");

var data = "";

switch (Convert.ToInt32(Console.ReadLine()))

{

case 1:

Console.WriteLine("Enter the data to insert:");

data = Console.ReadLine();

tree.InsertNode(Convert.ToInt32(data));

break;

case 2:

Console.WriteLine("Enter the data to remove:");

data = Console.ReadLine();

tree.RemoveNode(Convert.ToInt32(data));

break;

case 3:

Console.WriteLine("Enter the data to search in tree:");

data = Console.ReadLine();

var temp = tree.SearchNode(Convert.ToInt32(data));

Console.WriteLine(temp != null ? "Found!" : "Not found!");

break;

case 0:

Console.WriteLine("Terminating...");

return;

default:

Console.WriteLine("Try again");

break;

}

}

}

}

}

# **Приложение В. Сортировка вставками**

using System;

namespace InsertionSortApp

{

public static class Program

{

/// <summary>

/// Sort array

/// </summary>

/// <param name="array"> Unsorted array to sort </param>

/// <returns>

/// Returns array sorted in ascending

/// </returns>

private static int[] InsertionSort(int[] array)

{

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>

/// Method to swap two arguments

/// </summary>

/// <remarks>

/// Linked arguments required

/// </remarks>

/// <param name="valueA"> First value to swap </param>

/// <param name="valueB"> Second value to swap </param>

private static void Swap(ref int valueA, ref int valueB)

{

var temp = valueA;

valueA = valueB;

valueB = temp;

}

/// <summary>

/// Program's Main

/// </summary>

/// <param name="args"> Terminal args </param>

public static void Main(string[] args)

{

Console.Write("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)));

}

}

}