МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра вычислительной техники

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Основы разработки корпоративных систем на платформе .NET»

Тема: «Реализация базовых алгоритмов средствами языка C#»

Студент гр. 6305 _____ Стрельников В.Е.

Преподаватель _____ Пешехонов К.А.

Санкт-Петербург

2020

Содержание

Цель работы

Изучить основы программирования на языке C# в среде JetBrains Rider, научиться использовать Git, а также:

1. Получить практические навыки работы со связными списками;
2. Получить практические навыки работы с бинарными деревьями;
3. Получить практические навыки работы с алгоритмами сортировки.

Задание

1. Реализовать связный список: создание, удаление, добавление произвольных элементов, реверс списка - без использования стандартных коллекций/LINQ (только IEnumerable);
2. Реализовать бинарное дерево: заполнение, поиск, удаление элемента - без использования стандартных деревьев;
3. Реализовать сортировку вставками - без .OrderBy().

Выполнение

Связный список — структура данных, состоящая из элементов, содержащих помимо собственных данных ссылки на следующий и/или предыдущий элемент списка. Основные операции над связным список включают в себя добавление, удаление элемента, реверс списка. В данной лабораторной работе были реализованы следующий операции над двухсвязным списком:

1. Добавление элемента в начало списка;
2. Добавление элемента в конец списка;
3. Добавление элемента перед выбранным элементом списка;
4. Добавление элемента после выбранного элемента списка;
5. Удаление выбранного элемента списка;
6. Реверс списка;
7. Вывод списка.

Результат выполнения и компиляции программы с примерами

Для проверки работоспособности приложения добавим в список элементы (1,22,345,467). Затем выполним реверс списка, удалим элемент 345 и после элемента 1 вставим элемент 333. Выведем результат.

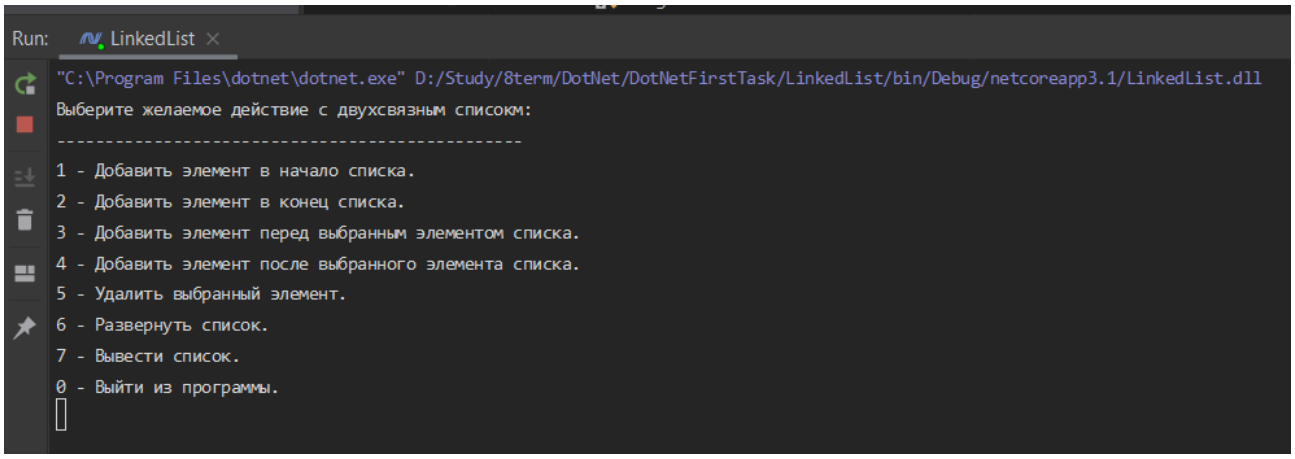

Рисунок 1. Запуск программы связного списка

```
Выберите желаемое действие с двухсвязным списокм:
-------------------------------------------------
1 - Добавить элемент в начало списка.
2 - Добавить элемент в конец списка.
3 - Добавить элемент перед выбранным элементом списка.
4 - Добавить элемент после выбранного элемента списка.
5 - Удалить выбранный элемент.
6 - Развернуть список.
7 - Вывести список.
0 - Выйти из программы.
7
DATA is: 467
DATA is: 345
DATA is: 22
DATA is: 1
```

Рисунок 2. Добавление элементов в список.

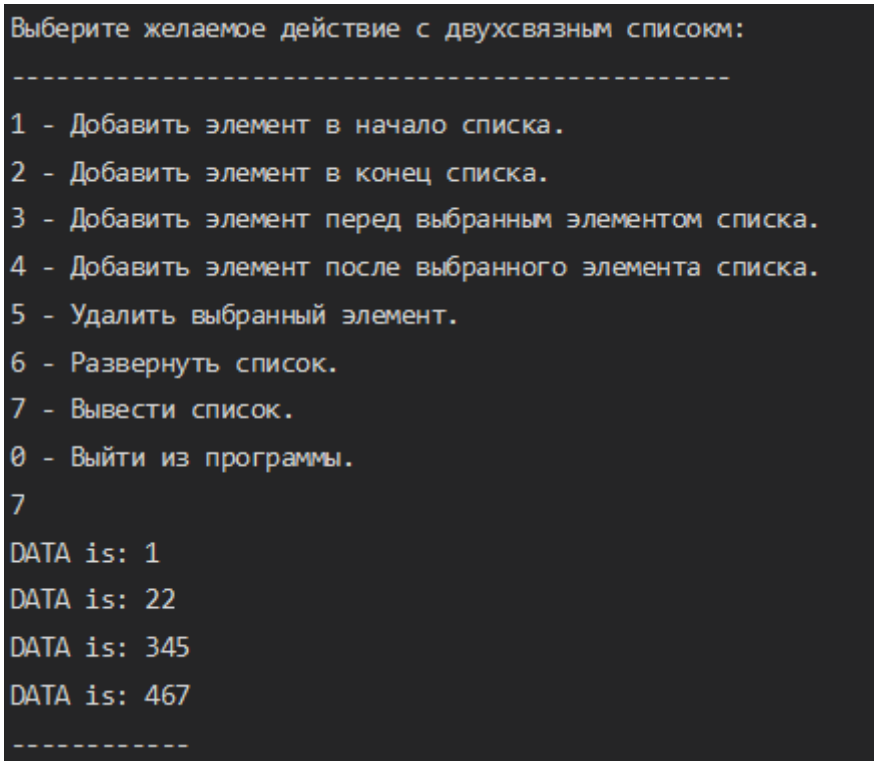

Рисунок 3. Реверс списка.

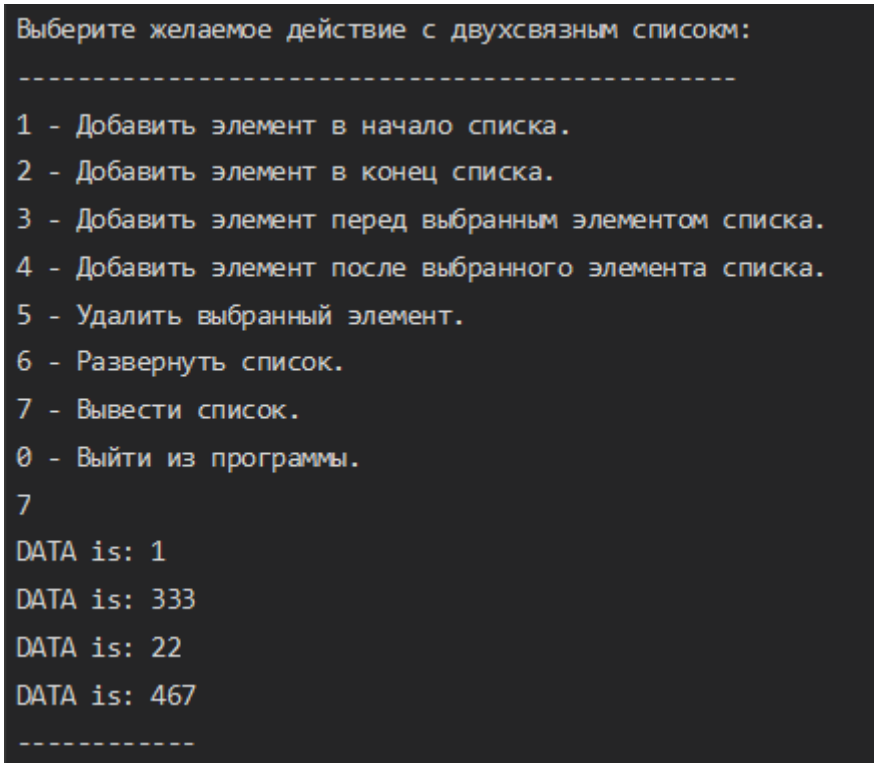

Рисунок 4. Удаление элемента 345, добавление элемента 333 после элемента 1.

Убеждаемся, что программа выполняет поставленную задачу верно и без ошибок.

Бинарное дерево — структура данных для работы с упорядоченными множествами.

Есть три операции обхода узлов дерева, отличающиеся порядком обхода узлов:

1. inorderTraversal — обход узлов в отсортированном порядке;
2. preorderTraversal — обход узлов в порядке: вершина, левое поддерево, правое поддерево;
3. postorderTraversal— обход узлов в порядке: левое поддерево, правое поддерево, вершина.

Также как и в связном списке основными операциями над деревом являются добавление и удаление элементов. В данной лабораторной работе были реализованы следующий операции над бинарным деревом:

1. Добавление элемента в дерево;
2. Удаление элемента из дерева;
3. Поиск элемента в дереве;
4. Прямой обход дерева;
5. Центрированный обход дерева;
6. Обратный обход дерева.

Результат выполнения и компиляции программы с примерами

Для проверки работоспособности приложения добавим в дерево элементы (1,6,8,7,2). Затем выполним все обходы дерева. Выполним поиск элемента 6, и элемента 4. Выведем результат.

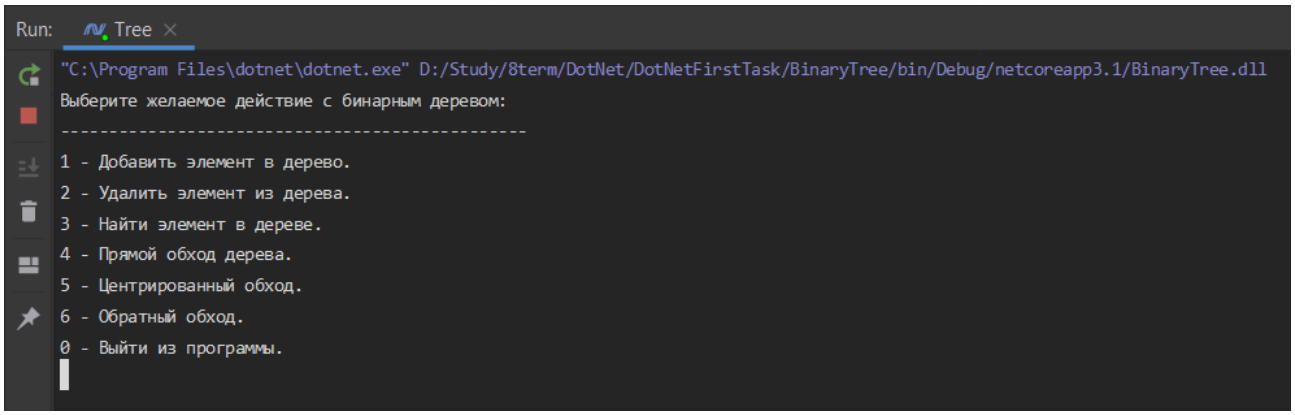

Рисунок 5. Запуск программы бинарного дерева

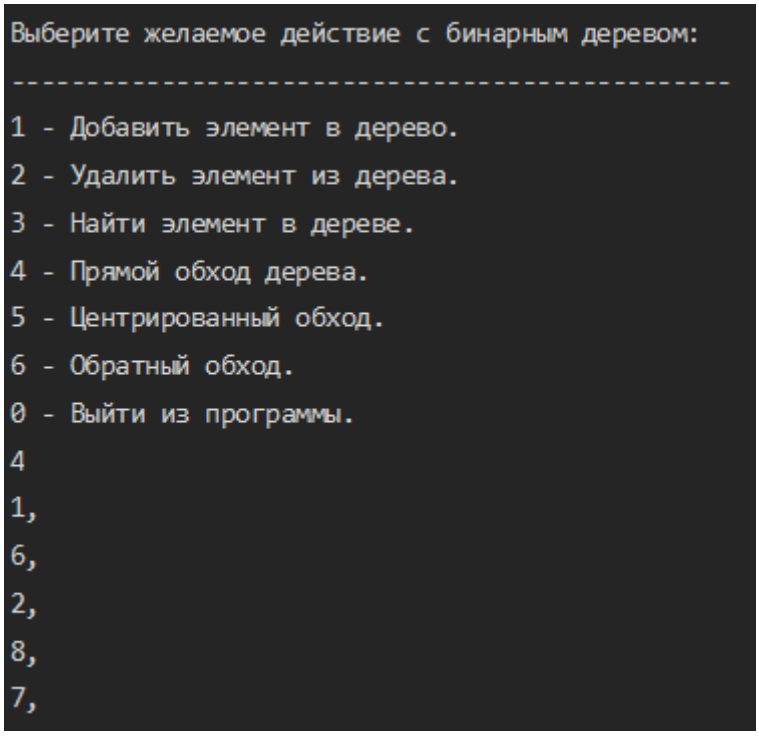

Рисунок 6. Результат прямого обхода дерева.

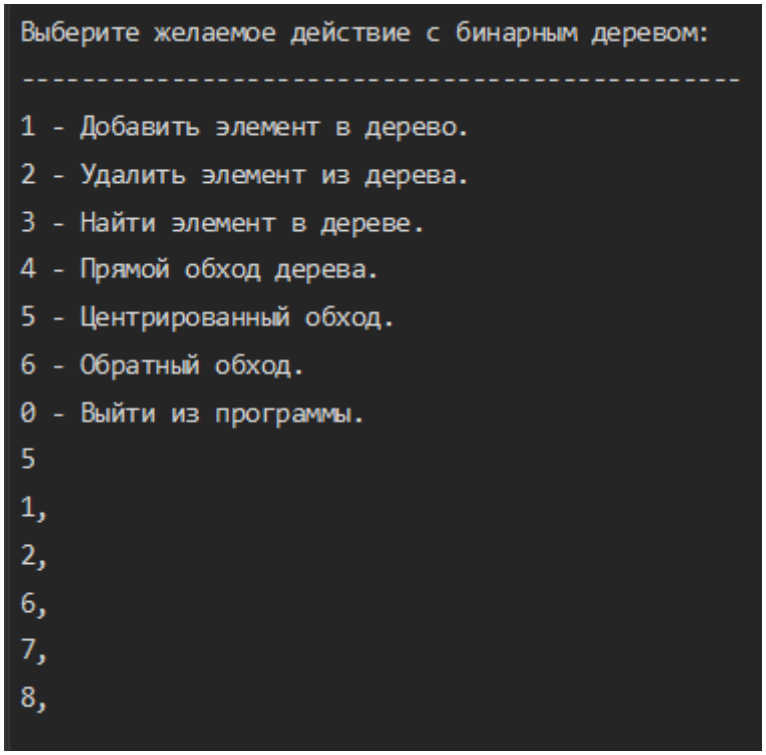

Рисунок 7. Результат центрированного обхода дерева.

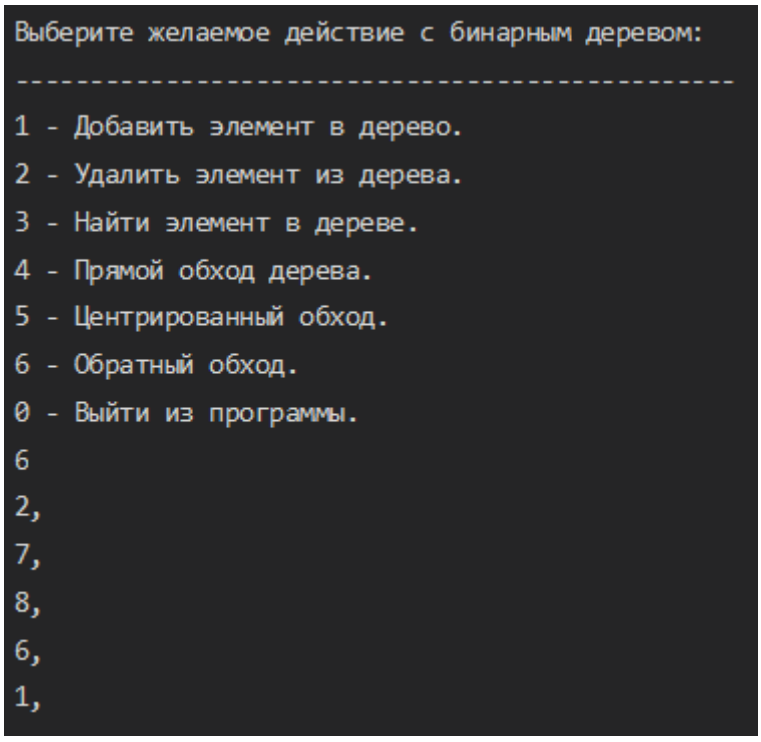

Рисунок 8. Результат обратного обхода дерева.

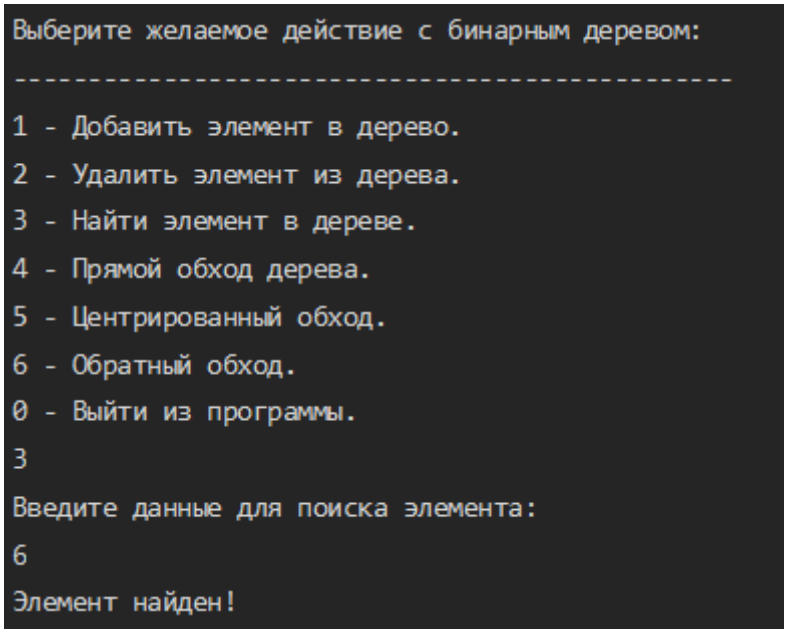

Рисунок 9. Результат поиска элемента 6.

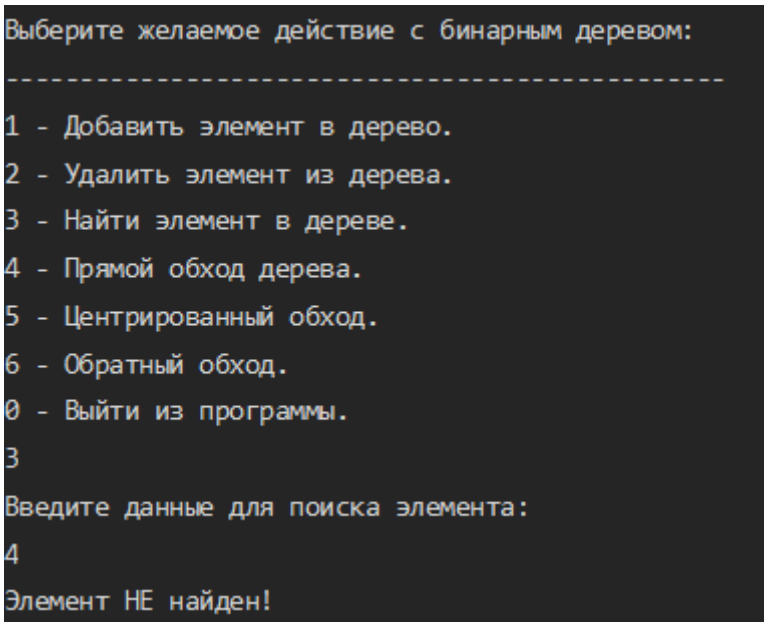

Рисунок 10. Результат поиска элемента 4.

Убеждаемся, что программа выполняет поставленную задачу верно и без ошибок.

Сортировка вставками — квадратичный алгоритм сортировки. Задача заключается в следующем: есть часть массива, которая уже отсортирована, и требуется вставить остальные элементы массива в отсортированную часть, сохранив при этом упорядоченность. Для этого на каждом шаге алгоритма мы выбираем один из элементов входных данных и вставляем его на нужную позицию в уже отсортированной части массива, до тех пор пока весь набор входных данных не будет отсортирован.

Результат выполнения и компиляции программы

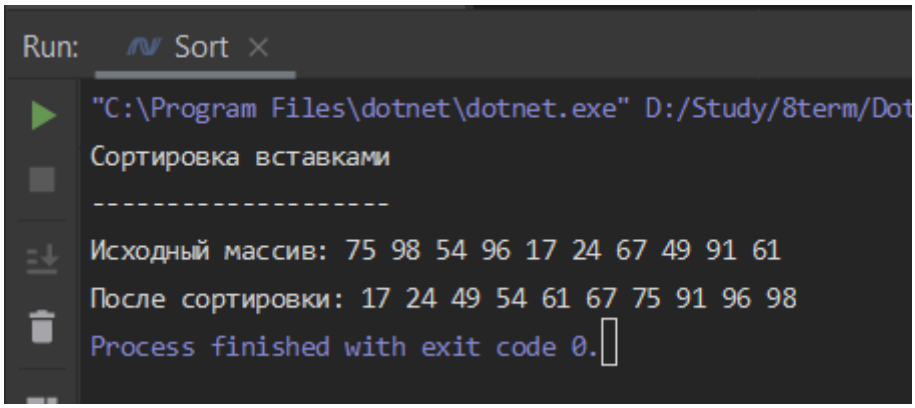

Рисунок 11. Результат выполнения сортировки вставками.

Вывод

В процессе выполнения данной лабораторной работы была установлена среда JetBrains Rider, создан репозиторий проекта на GitHub с использованием системы контроля версий Git, а также были реализованы связный список, бинарное дерево и алгоритм сортировки вставками на языке программирования C#.

Приложение

Linked list

```
using System;

namespace LinkedList
{
    public class Node
    {
        public string data;
```

```
    public Node next;
    public Node previous;

    public Node(string data)
    {
        this.data = data;
        next = null;
        previous = null;
    }
}

public class List
{
    public Node begin;

    public Node GetLast()
    {
        Node last = this.begin;
        while (last.next != null)
        {
            last = last.next;
        }

        return last;
    }

    /// <summary>
    /// Добавляет элемент в конец списка.
    /// </summary>
    /// <param name="data">Данные для элемента списка.</param>
    public void InsertAtEnd(string data)
    {
        Node node = new Node(data);
        if (this.begin == null)
        {
            node.previous = null;
            this.begin = node;
        }
        else
        {
            Node last = this.GetLast();
            last.next = node;
            node.previous = last;
        }
    }
    /// <summary>
    /// Добавляет элемент в начало списка.
    /// </summary>
    /// <param name="data">Данные для элемента списка.</param>
    public void InsertAtFront(string data)
    {
        Node node = new Node(data);
        node.next = this.begin;
        node.previous = null;
        if (this.begin != null)
        {
            this.begin.previous = node;
        }
        this.begin = node;
    }
    /// <summary>
    /// Добавляет элемент списка после указанного.
```

```
        /// </summary>
        /// <param name="selNumb">Номер элемента после которого необходимо
поместить элемент.</param>
        /// <param name="data">Данные для элемента списка.</param>
        public void InsertAfterSelected(int selNumb, string data)
        {
            Node selected = this.begin;
            int tempNum = 1;
            if (tempNum == selNumb)
            {
                selected = this.begin;
            }
            while (tempNum != selNumb)
            {
                selected = selected.next;
                tempNum += 1;
            }
            Node node = new Node(data);
            node.next = selected.next;
            selected.next = node;
            node.previous = selected;
            if (node.next != null)
            {
                node.next.previous = node;
            }
        }

        /// <summary>
        /// Добавление элемента списка перед указанным.
        /// </summary>
        /// <param name="selNumb">Номер элемента перед которым необходимо
поместить элемент.</param>
        /// <param name="data">Данные для элемента списка.</param>
        public void InsertBeforeSelected(int selNumb, string data)
        {
            Node selected = this.begin;
            int tempNum = 1;
            if (tempNum == selNumb)
            {
                this.InsertAtFront(data);
                return;
            }
            while (tempNum != selNumb)
            {
                selected = selected.next;
                tempNum += 1;
            }
            Node node = new Node(data);
            node.next = selected;
            node.previous = selected.previous;

            selected.previous.next = node;
            selected.previous = node;

            if (node.next != null)
            {
                node.next.previous = node;
            }
        }
        /// <summary>
        /// Удаление выбранного элемента списка.
        /// </summary>
```

```
/// <param name="delNumber">Номер удаляемого элемента.</param>
public void DeleteSelected(int delNumber)
{
    Node node = this.begin;
    int tempNum = 1;
    if (tempNum == delNumber)
    {
        this.begin = node.next;
        node.next.previous = node.previous;
    }
    while (tempNum != delNumber)
    {
        node = node.next;
        tempNum += 1;
    }
    if (node == null)
    {
        return;
    }
    if (node.next != null)
    {
        node.next.previous = node.previous;
    }
    if (node.previous != null)
    {
        node.previous.next = node.next;
    }
}
/// <summary>
/// Разворот списка.
/// </summary>
public void ListReverse()
{
    if (this.begin == null || this.begin.next == null)
    {
        return;
    }

    Node current = this.begin;

    while (current.next != null)
    {
        current = current.next;
    }

    this.begin = current;

    while (current != null)
    {
        Node prev = current.previous;
        current.previous = current.next;
        current.next = prev;
        current = prev;
    }
}
/// <summary>
/// Вывод данных элементов списка.
/// </summary>
public void PrintList()
{
    if (this.begin == null)
    {
```

```
                Console.WriteLine("Список пуст!");
                return;
            }
            Node here = this.begin;
            Console.WriteLine("DATA is: {0}",here.data);
            while (here.next != null)
            {
                here = here.next;
                Console.WriteLine("DATA is: {0}",here.data);
            }
            Console.WriteLine("------------");
        }
    }

    class Program
    {
        private static void Main()
        {
            var list = new List();
            var exit = false;
            while (!exit)
            {
                Console.WriteLine("Выберите желаемое действие с двухсвязным
списокм:");
                Console.WriteLine("------------------------------------------
---");
                Console.WriteLine("1 - Добавить элемент в начало списка.");
                Console.WriteLine("2 - Добавить элемент в конец списка.");
                Console.WriteLine("3 - Добавить элемент перед выбранным
элементом списка.");
                Console.WriteLine("4 - Добавить элемент после выбранного
элемента списка.");
                Console.WriteLine("5 - Удалить выбранный элемент.");
                Console.WriteLine("6 - Развернуть список.");
                Console.WriteLine("7 - Вывести список.");
                Console.WriteLine("0 - Выйти из программы.");
                var val = Console.ReadLine();
                var choice = Convert.ToInt32(val);
                var data = "";
                int key;
                switch (choice)
                {
                    case 1:
                        Console.WriteLine("Введите данные для добавляемого
элемента:");
                        data = Console.ReadLine();
                        list.InsertAtFront(data);
                        break;
                    case 2:
                        Console.WriteLine("Введите данные для добавляемого
элемента:");
                        data = Console.ReadLine();
                        list.InsertAtEnd(data);
                        break;
                    case 3:
                        Console.WriteLine("Введите номер элемента:");
                        data = Console.ReadLine();
                        key = Convert.ToInt32(data);
                        Console.WriteLine("Введите данные для добавляемого
элемента:");
                        data = Console.ReadLine();
                        list.InsertBeforeSelected(key,data);
```

```
                        break;
                    case 4:
                        Console.WriteLine("Введите номер элемента:");
                        data = Console.ReadLine();
                        key = Convert.ToInt32(data);
                        Console.WriteLine("Введите данные для добавляемого
элемента:");
                        data = Console.ReadLine();
                        list.InsertAfterSelected(key,data);
                        break;
                    case 5:
                        Console.WriteLine("Введите номер удаляемого элемента:");
                        data = Console.ReadLine();
                        var delChoice = Convert.ToInt32(data);
                        list.DeleteSelected(delChoice);
                        break;
                    case 6:
                        list.ListReverse();
                        break;
                    case 7:
                        list.PrintList();
                        break;
                    case 0:
                        exit = true;
                        break;
                    default:
                        Console.WriteLine("Error.");
                        break;
                }

            }

        }
    }
}
```

Binary tree

```
using System;

namespace BinaryTree
{
    class Node
    {
        public int value;
        public Node left;
        public Node right;
    }

    class BinTree
    {
        public Node root = null;

        /// <summary>
        /// Добавление элемента в дерево
        /// </summary>
        /// <param name="key"></param>
        /// <param name="leaf"></param>
        public void insertNode(int key, Node leaf)
        {

            if (key < leaf.value)
```

```
        {
            if (leaf.left != null)
            {
                insertNode(key, leaf.left);
            }
            else
            {
                leaf.left = new Node();
                leaf.left.value = key;
                leaf.left.left = null;
                leaf.left.right = null;
            }
        }
        else if (key >= leaf.value)
        {
            if (leaf.right != null)
            {
                insertNode(key, leaf.right);
            }
            else
            {
                leaf.right = new Node();
                leaf.right.value = key;
                leaf.right.right = null;
                leaf.right.left = null;
            }
        }
    }

    public void insertNode(int key)
    {
        if (this.root != null)
        {
            insertNode(key, this.root);
        }
        else
        {
            this.root = new Node();
            this.root.value = key;
            this.root.left = null;
            this.root.right = null;
        }
    }

    public Node findMinimum(Node cur)
    {
        while (cur.left != null)
        {
            cur = cur.left;
        }

        return cur;
    }
    /// <summary>
    /// Поиск элемента в дереве
    /// </summary>
    /// <param name="key"></param>
    /// <param name="leaf"></param>
    /// <returns></returns>
    public Node searchNode(int key, Node leaf)
    {
        if (leaf != null)
```

```
    {
        if (key == leaf.value)
        {
            return leaf;
        }

        if (key < leaf.value)
        {
            return searchNode(key, leaf.left);
        }
        else
        {
            return searchNode(key, leaf.right);
        }
    }
    else
    {
        return null;
    }
}

public Node searchNode(int key)
{
    return searchNode(key, this.root);
}

/// <summary>
/// Удаление элемента в дереве
/// </summary>
/// <param name="root"></param>
/// <param name="deleteNode"></param>
/// <returns></returns>
public Node deleteN(Node root, Node deleteNode)
{
    if (root == null)
    {
        return root;
    }

    if (deleteNode.value < root.value)
    {
        root.left = deleteN(root.left, deleteNode);
    }

    if (deleteNode.value > root.value)
    {
        root.right = deleteN(root.right, deleteNode);
    }

    if (deleteNode.value == root.value)
    {
        if (root.left == null && root.right == null)
        {
            root = null;
            return root;
        }

        else if (root.left == null)
        {
            Node temp = root;
            root = root.right;
            temp = null;
```

```
        }

        else if (root.right == null)
        {
            Node temp = root;
            root = root.left;
            temp = null;
        }

        else
        {
            Node min = findMinimum(root.right);
            root.value = min.value;
            root.right = deleteN(root.right, min);
        }
    }

    return root;
}

public void deleteN(int key)
{
    Node deleteNode = searchNode(key);
    deleteN(root, deleteNode);
}

/// <summary>
/// Прямой обход дерева
/// </summary>
public void preorder_print()
{
    preorder_print(this.root);
    Console.WriteLine("");
}

public void preorder_print(Node leaf)
{
    if (leaf != null)
    {
        Console.WriteLine("{0},", leaf.value);
        preorder_print(leaf.left);
        preorder_print(leaf.right);
    }
}


/// <summary>
/// Центрированный обход дерева
/// </summary>
public void inorder_print()
{
    inorder_print(this.root);
    Console.WriteLine("");
}

public void inorder_print(Node leaf)
{
    if (leaf != null)
    {
        inorder_print(leaf.left);
        Console.WriteLine("{0},", leaf.value);
        inorder_print(leaf.right);
```

```
            }
        }

        /// <summary>
        /// Обратный обход дерева
        /// </summary>
        public void postorder_print()
        {
            postorder_print(this.root);
            Console.WriteLine("");
        }

        public void postorder_print(Node leaf)
        {
            if (leaf != null)
            {
                postorder_print(leaf.left);
                postorder_print(leaf.right);
                Console.WriteLine("{0},", leaf.value);
            }
        }

    }

    class Program
    {
        static void Main(string[] args)
        {
            BinTree tree = new BinTree();
            var exit = false;
            Node temp;
            while (!exit)
            {
                Console.WriteLine("Выберите желаемое действие с бинарным
деревом:");
                Console.WriteLine("-----------------------------------------
---");
                Console.WriteLine("1 - Добавить элемент в дерево.");
                Console.WriteLine("2 - Удалить элемент из дерева.");
                Console.WriteLine("3 - Найти элемент в дереве.");
                Console.WriteLine("4 - Прямой обход дерева.");
                Console.WriteLine("5 - Центрированный обход.");
                Console.WriteLine("6 - Обратный обход.");
                Console.WriteLine("0 - Выйти из программы.");
                var val = Console.ReadLine();
                var choice = Convert.ToInt32(val);
                var data = "";
                int key;
                switch (choice)
                {
                    case 1:
                        Console.WriteLine("Введите данные для добавляемого
элемента:");
                        data = Console.ReadLine();
                        tree.insertNode(Convert.ToInt32(data));
                        break;
                    case 2:
                        Console.WriteLine("Введите данные для удаляемого
элемента:");
                        data = Console.ReadLine();
                        tree.deleteN(Convert.ToInt32(data));
                        break;
```

```
                    case 3:
                        Console.WriteLine("Введите данные для поиска
элемента:");
                        data = Console.ReadLine();
                        temp = tree.searchNode(Convert.ToInt32(data));
                        if (temp != null)
                        {
                            Console.WriteLine("Элемент найден!");
                        }
                        else
                        {
                            Console.WriteLine("Элемент НЕ найден!");
                        }
                        break;
                    case 4:
                        tree.preorder_print();
                        break;
                    case 5:
                        tree.inorder_print();
                        break;
                    case 6:
                        tree.postorder_print();
                        break;
                    case 0:
                        exit = true;
                        break;
                    default:
                        Console.WriteLine("Error.");
                        break;
                }

            }
        }
    }
}
```

Insertion sort

```
using System;

namespace InsertSort
{
    class Program
    {
        static void Main(string[] args)
        {
            const int Min = 0;
            const int Max = 100;
            var array = new int[10];

            var randNum = new Random();
            for (var i = 0; i < array.Length; i++)
            {
                array[i] = randNum.Next(Min, Max);
            }
            Console.WriteLine("Сортировка вставками");
            Console.WriteLine("--------------------");
            Console.Write("Исходный массив: ");
            foreach (var t in array)
            {
                Console.Write(t + " ");
            }
```

```
            for (var i = 1; i < array.Length; i++)
            {
                var value = array[i];
                var flag = false;
                for (var j = i - 1; j >= 0 && flag != true; )
                {
                    if (value < array[j])
                    {
                        array[j + 1] = array[j];
                        j--;
                        array[j + 1] = value;
                    }
                    else
                    {
                        flag = true;
                    }
                }
            }
            Console.WriteLine("");
            Console.Write("После сортировки: ");
            foreach (var t in array)
            {
                Console.Write(t + " ");
            }
        }
    }
}
```