Рекурсивные структуры данных

Примеры программ

1. Списки

Написать класс, реализующий односвязный список. Класс должен содержать методы `Push` (добавление элемента в начало списка), `Pop` (извлечение элемента из начала списка). Написать:

- метод `Print`, выводящий список в виде строки;
- метод `Sum`, вычисляющий сумму элементов списка;
- метод `Remove`, удаляющий элемент из списка; и если удаление прошло успешно, то возвращает `true`, иначе – `false`.

Написать программу, использующую этот класс.

Файл `MyLinkedList.cs`

```
using System;
using System.Text;

namespace MyLinkedList
{
    public class ListUnderflow : Exception { }

    public class LinkedList
    {
        class Node
        {
            public double value;
            public Node next;
        }
        Node head = null;

        public void Push(double x)
        {
            Node n = new Node();
            n.value = x;
            n.next = head;
            head = n;
        }

        public double Pop()
        {
            if (head == null)
                throw new ListUnderflow();
            double val = head.value;
            head = head.next;
            return val;
        }

        public void Print()
        {
            if (head == null)
                throw new ListUnderflow();
            StringBuilder PrintList = new StringBuilder();
            Node temp = head;
            while (temp != null)
            {
                PrintList.Append(temp.value);
                PrintList.Append(" ");
                temp = temp.next;
            }
            Console.WriteLine(PrintList);
```

```
        }

        public double Sum()
        {
            if (head == null)
                throw new ListUnderflow();
            double S = 0;
            Node temp = head;
            while (temp != null)
            {
                S += temp.value;
                temp = temp.next;
            }
            return S;
        }

        public bool Remove(double x)
        {
            if (head == null)
                throw new ListUnderflow();
            if (head.value == x)
            {
                Pop();
                return true;
            }

            Node temp = head;
            while (temp.next.next != null && temp.next.value != x)
            {
                temp = temp.next;
            }
            if (temp.next.value == x)
            {
                if (temp.next.next != null )
                {
                    temp.next = temp.next.next;
                }
                else
                {
                    temp.next = null;
                }
                return true;
            }
            return false;
        }

    }
}
```

Файл Program.cs

```
using System;

namespace MyLinkedList
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                int a;
                LinkedList l = new LinkedList();
                l.Push(1);
```

```
                l.Push(3);
                l.Push(4);
                l.Push(5);
                l.Push(7);

                Console.Write("Список: ");
                l.Print();
                Console.WriteLine("Сумма элементов = {0}", l.Sum());
                Console.WriteLine("Введите элемент для удаления");
                a = Convert.ToInt32(Console.ReadLine());
                if (l.Remove(a)) Console.WriteLine("Элемент {0} удален из списка", a);
                else Console.WriteLine("Элемент {0} в списке отсутствует", a);
                Console.WriteLine("Удален элемент {0} из начала списка", l.Pop());
                Console.Write("Список: ");
                l.Print();
                Console.ReadLine();
            }
            catch (ListUnderflow)
            {
                Console.WriteLine("Список пуст");
            }
            Console.ReadLine();
        }
    }
}
```

Результат выполнения программы:

```
Список: 7 5 4 3 1
Сумма элементов = 20
Введите элемент для удаления
4
Элемент 4 удален из списка
Удален элемент 7 из начала списка
Список: 5 3 1
```

```
Список: 7 5 4 3 1
Сумма элементов = 20
Введите элемент для удаления
10
Элемент 10 в списке отсутствует
Удален элемент 7 из начала списка
Список: 5 4 3 1
```

2. Деревья

Написать класс, реализующий бинарное дерево поиска. Класс должен содержать методы `Add` (добавление элемента в дерево), `InOrderWalk` (симметричный обход дерева). Написать метод `Sum`, вычисляющий сумму элементов дерева. Написать программу, использующую этот класс.

Файл `Trees.cs`

```
using System.Collections.Generic;

namespace BinaryTrees
{
    /// <summary>
    /// Дерево сортировки
    /// </summary>
    public class Tree
    {
        /// <summary>
        /// Узел дерева
        /// </summary>
        class Node
        {
            public Node Left, Right;    // Ссылки на левое и правое поддеревья
            public double Value;        // Хранимое в узле значение
```

```
    /// <summary>
    /// Создаёт узел дерева
    /// </summary>
    /// <param name="val">Хранимое в узле значение</param>
    public Node(double val)
    {
        Left = Right = null;
        Value = val;
    }
}

Node root = null;   // Ссылка на корень дерева

/// <summary>
/// Добавляет в дерево сортировки значение
/// </summary>
/// <param name="val">Добавляемое значение</param>
public void Add(double val)
{
    if (root == null)
    {
        root = new Node(val);
        return;
    }

    var x = root;   // Ссылка на текущий узел
    while (true)
        if (val <= x.Value)
        {
            if (x.Left == null)
            {
                x.Left = new Node(val);
                return;
            }
            else
                x = x.Left;
        }
        else {
            if (x.Right == null)
            {
                x.Right = new Node(val);
                return;
            }
            else
                x = x.Right;
        }
}

/// <summary>
/// Симметричный обход всего дерева
/// </summary>
/// <returns>Список элементов</returns>
public List<double> InOrderWalk() {
    return InOrderWalk(root);
}

/// <summary>
/// Симметричный обход поддерева t
/// </summary>
/// <param name="t">Ссылка на поддерево</param>
/// <returns>Список элементов</returns>
List<double> InOrderWalk(Node t)
{
    if (t == null)
        return new List<double>();

    var L = InOrderWalk(t.Left);
```

```
            L.Add(t.Value);
            L.AddRange(InOrderWalk(t.Right));

            return L;
        }

        // сумма элементов дерева
        public double Sum()
        {
            double S = 0;
            SumTree(root, ref S);
            return S;
        }

        void SumTree(Node t, ref double S)
        {
            if (t != null)
            {
                S += t.Value;
                SumTree(t.Left, ref S);
                SumTree(t.Right, ref S);
            }
        }

    }
}
```

Файл Program.cs

```
using System;

namespace BinaryTrees
{
    class Program
    {
        static void Main(string[] args)
        {
            Tree t = new Tree();

            t.Add(4);
            t.Add(2);
            t.Add(1);
            t.Add(3);
            t.Add(6);
            t.Add(5);
            t.Add(7);

            Console.WriteLine(string.Join(", ", t.InOrderWalk()));
            Console.WriteLine("Сумма элементов дерева: " + t.Sum());

        }
    }
}
```

Результат выполнения программы: