Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Ярославский государственный технический университет»
Кафедра «Информационные системы и технологии»

Проект защищен
с оценкой _____
Руководитель
_____ А.Н. Прозоров
«___»_____ 2025

РАЗРАБОТКА MDI-ПРИЛОЖЕНИЯ С ИСПОЛЬЗОВАНИЕМ WINDOWS FORMS ДЛЯ РЕДАКТОРА ФАЙЛОВЫХ ИЕРАРХИЧЕСКИХ СТРУКТУР

Расчетно-пояснительная записка к курсовому проекту
по дисциплине «Технологии программирования»

ЯГТУ 09.03.02-010 КП

Нормоконтролер
Преподаватель
_____А.Н. Прозоров
«___»_____2025

Проект выполнила
студентка группы ЗЦИС-26
_____ Ю.С. Липенко
«___»_____ 2025

2025

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Ярославский государственный технический университет» Кафедра Информационные системы и технологии

ЗАДАНИЕ №10
ПО КУРСОВОМУ ПРОЕКТИРОВАНИЮ

Студенту Липенко Ю.С.. факультет заочный курс 2 группа ЗЦИС-26

Тема проекта и исходные данные
Разработка приложения редактор файловых иерархических структур в C# с использованием Windows Forms MDI.

2 Представить следующие материалы
1) текстовые
а)_____
б)_____
в)_____
г)_____
д)_____
е)_____

2) графические
а)_____
б)_____
в)_____

3 Рекомендуемая литература и материалы
1. Троелсен и Джепикс. Язык программирования C#
2. Скит. C# для профессионалов. Тонкости программирования
3. Документация по C# - https://docs.microsoft.com/ru-ru/dotnet/csharp/

4 Дата выдачи задания_____
5 Срок сдачи законченного проекта_____
6 Отметка о явке на консультацию:
1)_____2)_____3)_____
4)_____5)_____6)_____

Руководитель проекта
Прозоров А.Н.

Зав кафедрой
Бойков С.Ю.

Содержание

1 Введение

В современном мире организация и управление данными играют ключевую роль. Файловые системы, представляющие собой иерархические структуры, являются основой хранения и доступа к информации на компьютерах.

Данный курсовой проект посвящен разработке приложения, реализующего функции редактора иерархических файловых структур.

Целью проекта является создание многодокументного интерфейса (MDI) приложения на языке C# с использованием Windows Forms, которое позволит пользователям визуализировать, создавать, редактировать и управлять файловой системой. Приложение предоставит инструменты для выполнения основных операций с файлами и каталогами, таких как открытие, создание, удаление, копирование, а также для навигации по файловой системе и запуска исполняемых файлов.

В качестве основного элемента пользовательского интерфейса будет использоваться компонент TreeView, обеспечивающий наглядное представление иерархической структуры данных.

2. Описание задания

Создать в C# с использованием Windows Forms MDI приложение – редактор файловых иерархических структур, позволяющий открывать, создавать и редактировать деревья файловой системы, переходить по каталогам, запускать исполнимые файлы, удалять каталоги и файлы, копировать файлы, сохранять иерархические структуры в файлах, открывать иерархические структуры из файлов. Для реализации необходимо использовать класс TreeView.

3. Подробное описание используемых алгоритмов и классов

Разрабатываемое приложение представляет собой MDI-редактор, то есть интерфейс нескольких документов, предназначенный для создания, редактирования и сохранения иерархических структур. Которые в свою очередь отображаются в виде дерева каталогов (TreeView).

Архитектура приложения «Редактор файловых иерархических структур» состоит из двух форм: Form1 главной и ChildForm дочерней.

На Form1 размещено меню с такими пунктами, как «Файл», «Правка», «Вид».

Через пункт «Файл», подпункт «Создать» вызывается дочернее окно, ChildForm, задачей которого является отображение иерархической структуры в виде элемента управления TreeView.

В приложении предусмотрены следующие возможности для работы:

- Создание нового файла иерархии с нуля.
- Открытие существующего файла иерархии для просмотра и редактирования. Поддерживается открытие файлов с расширением .hier.
- Визуализация иерархической структуры в виде дерева каталогов TreeView.
- Отображение корневых дисков при запуске приложения.
- Динамическая загрузка содержимого папок при развертывании узлов дерева.
- Добавление новых дочерних узлов к уже существующим.
- Удаление выбранных узлов из дерева.
- Переименование узлов.
- Копирование и вставка узлов в другие места в дереве или в другие экземпляры приложения.
- Вырезание узлов с последующей вставкой.
- Развернуть все узлы в дереве, чтобы увидеть всю структуру.
- Свернуть все узлы в дереве, чтобы отображались только корневые элементы.
- Обновить представление дерева, перезагрузив структуру дисков и папок.
- Сохранить текущую иерархическую структуру в файле.
- Сохранить текущую иерархическую структуру в новый файл, позволяя выбрать имя и местоположение файла.
- Поддержка открытия нескольких файлов иерархии одновременно в отдельных дочерних окнах.

Для реализации описанных выше возможностей были использованы следующие классы и алгоритмы:

Класс ChildForm является ключевым компонентом приложения, поскольку он отвечает за отображение, редактирование и управление

иерархической структурой. Его основное назначение — предоставить возможность визуализировать иерархические структуры, представленные в виде дерева каталогов, и управлять ими.

1. Алгоритм обхода дерева

Для сохранения иерархической структуры из TreeView в файл используется рекурсивный алгоритм обхода дерева. Этот алгоритм эффективно обрабатывает иерархию любой глубины. Алгоритм работает следующим образом:

1. Начинается с корневых узлов TreeView, находящихся непосредственно в коллекции treeView1.Nodes.
2. Для каждого узла:
   2.1. Текст текущего узла сохраняется в список строк.
   2.2. Для каждого дочернего узла текущего узла этот же алгоритм вызывается рекурсивно.

2. Алгоритм обработки события AfterLabelEdit позволяет изменить название узла в TreeView.

3. Для LoadHierarchyFromFile :

1. Считываются все строки из файла.
2. Создается цикл по каждой строке.
3. Определяется уровень вложенности узла.
4. Создается новый TreeNode с текстом из строки.

Если уровень вложенности текущего узла больше, чем уровень вложенности предыдущего узла, добавляется текущий узел как дочерний элемент к предыдущему узлу.

В противном случае добавляется текущий узел в качестве дочернего элемента к родительскому узлу предыдущего узла.

4 Инструкция пользователя по работе с программой

1. Запуск программы:

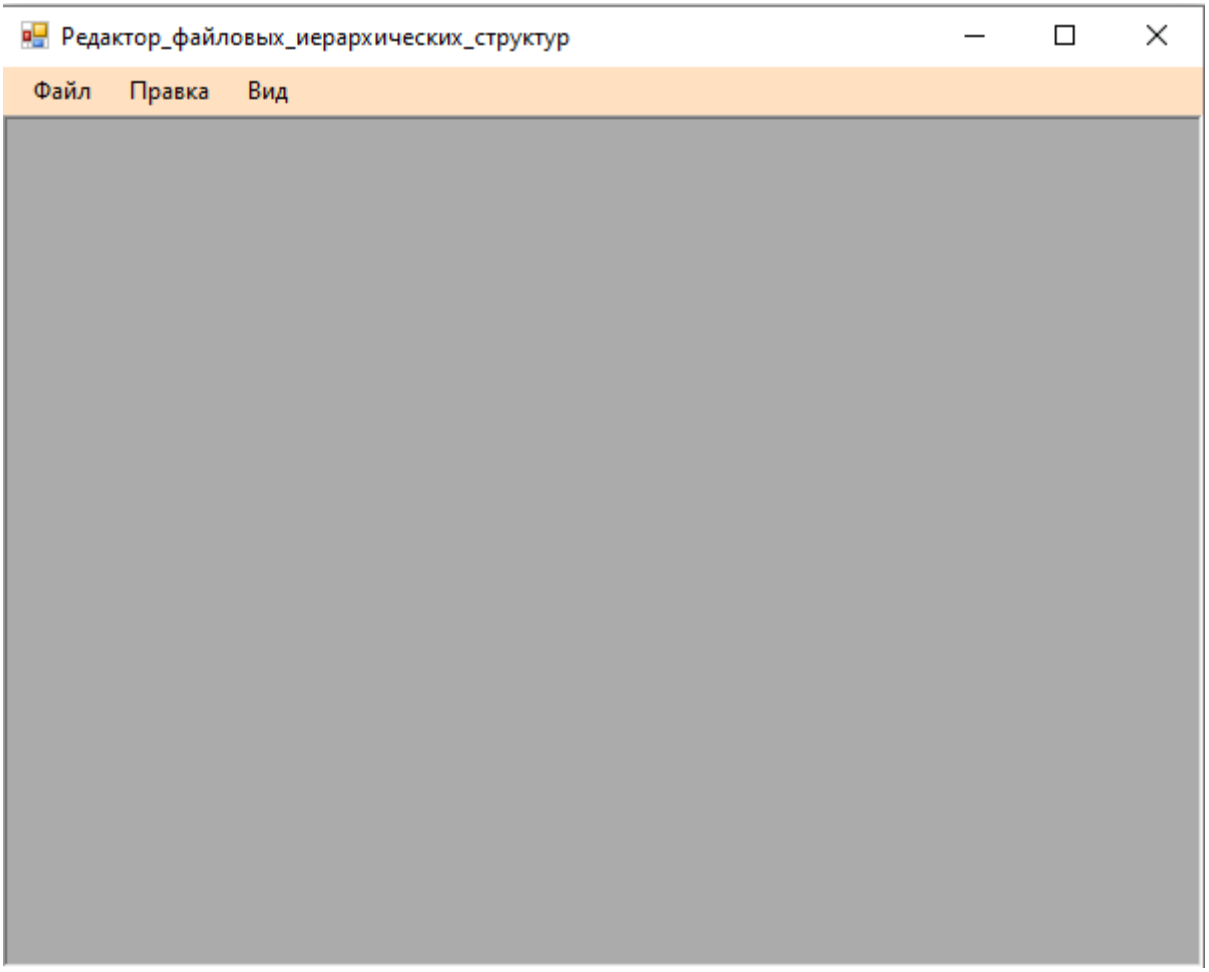

Рисунок 1. Главное окно программы Form1

2. Для начала работы необходимо выбрать пункт меню «Файл», подпункт «Создать».

Программа отображает файловую систему в виде дерева в левой части окна.

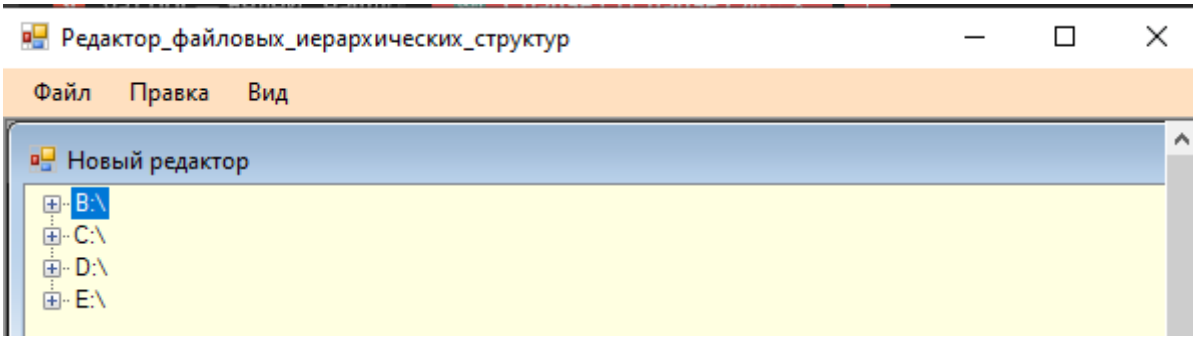

Рисунок 2. Новый редактор, файловая система в виде дерева

3. Для разворачивания или сворачивания окна можно воспользоваться пунктом меню «Вид», подпункт «Развернуть все» или «Свернуть все» соответственно, либо при помощи нажатия левой кнопкой мыши по знакам «плюс» или «минус».

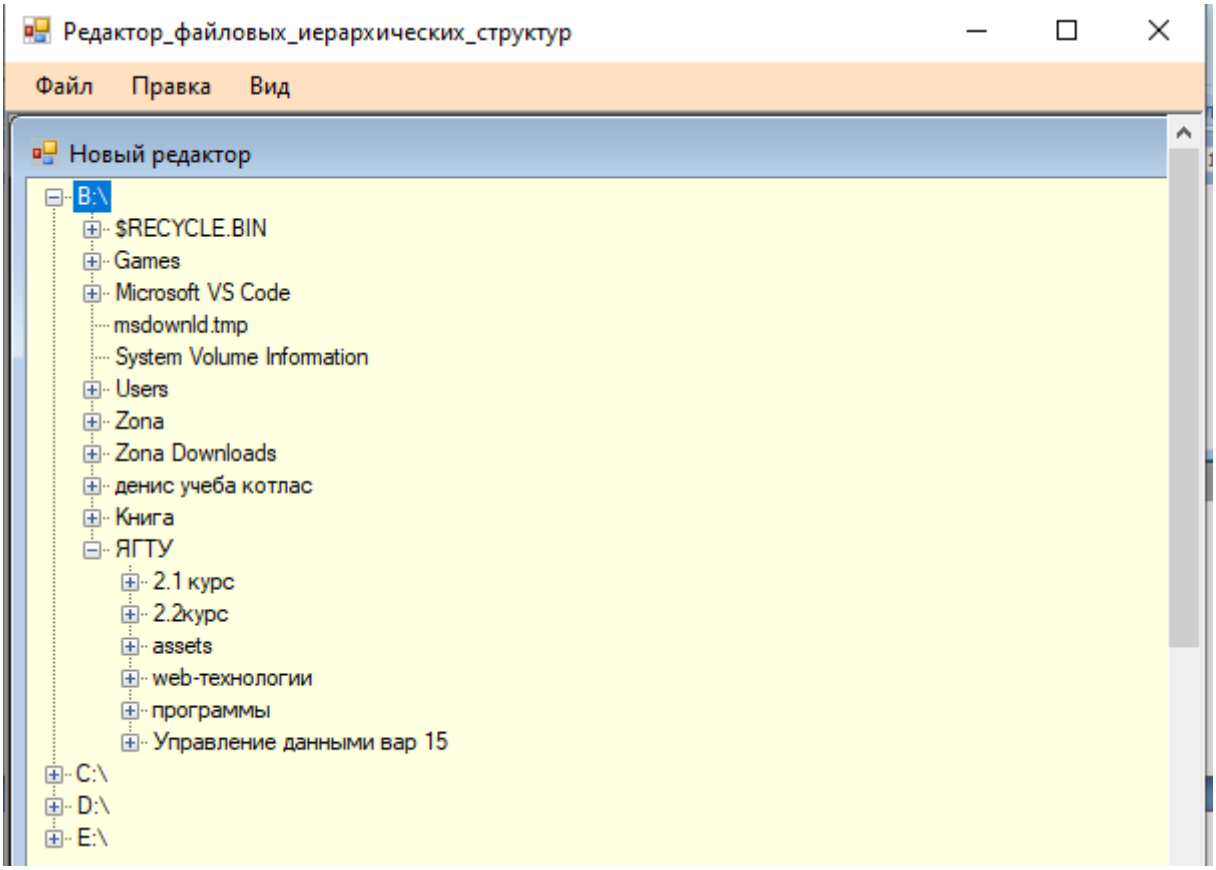

Рисунок 3. Взаимодействие с файловой системой

4. Для выполнения операций копировать, вставить, вырезать, удалить, переименовать, добавить с папками и файлами необходимо выбрать требуемый элемент, после чего воспользоваться пунктом меню «Правка», подпункт «Копировать», «Вставить», «Вырезать», «Удалить», «Переименовать», «Добавить» соответственно, либо аналогичным образом нажав правой кнопкой мыши по элементу и выбрав пункты контекстного меню.

Рисунок 4. Добавленная папка

5. Для отмены последнего действия или его возвращения необходимо воспользоваться кнопками на соответствующей Новой форме обозначенными знаками больше «<» и меньше «>».

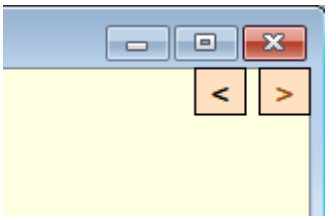

Рисунок 5. Кнопки отмены, возвращения действия

6. Для сохранения текущей структуры файловой системы в файл *.hier необходимо нажать в меню пункте «Файл» подпункт «Сохранить».

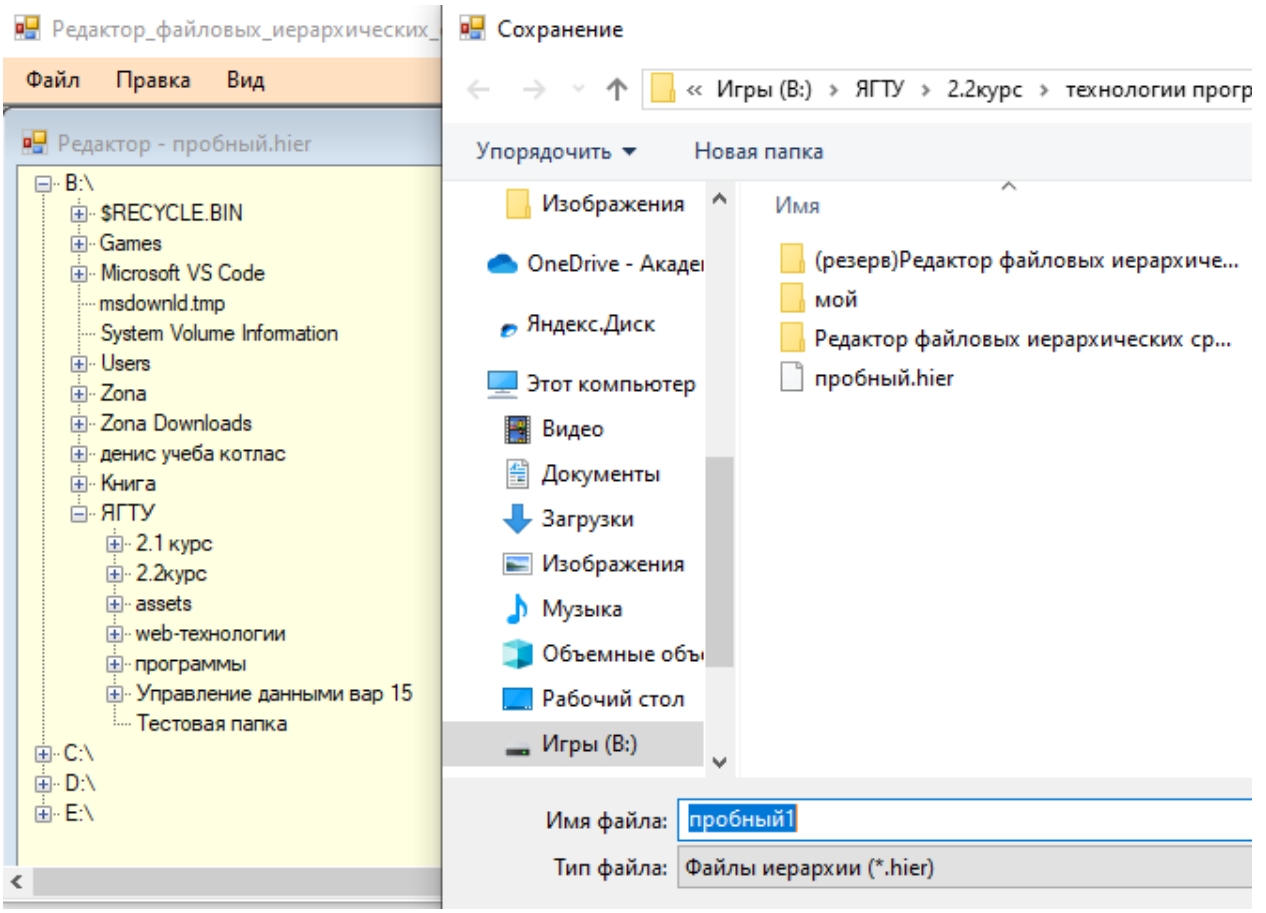

Рисунок 6. Сохранение файловой структуры

7. Для открытия файла *.hier необходимо нажать в меню, пункте «Файл» подпункт «Открыть» и выбрать требуемый файл.

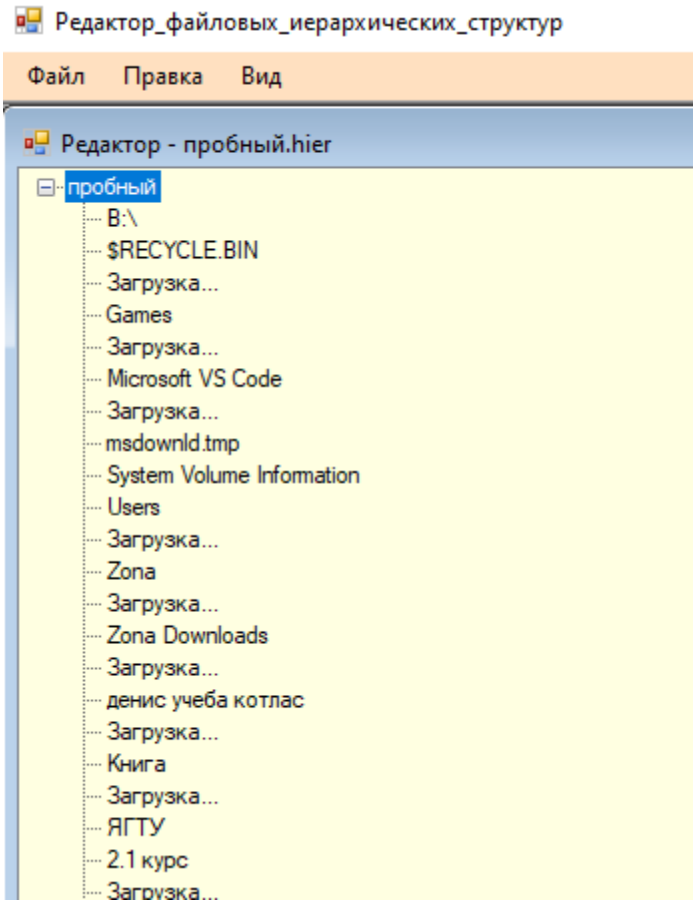

Рисунок 7. Открытый сохраненный ранее файл

8. При необходимости одновременной работы можно открыть одновременно несколько дочерних окно и расположить их в главном окне, перетаскивая зажатием левой кнопки мыши.

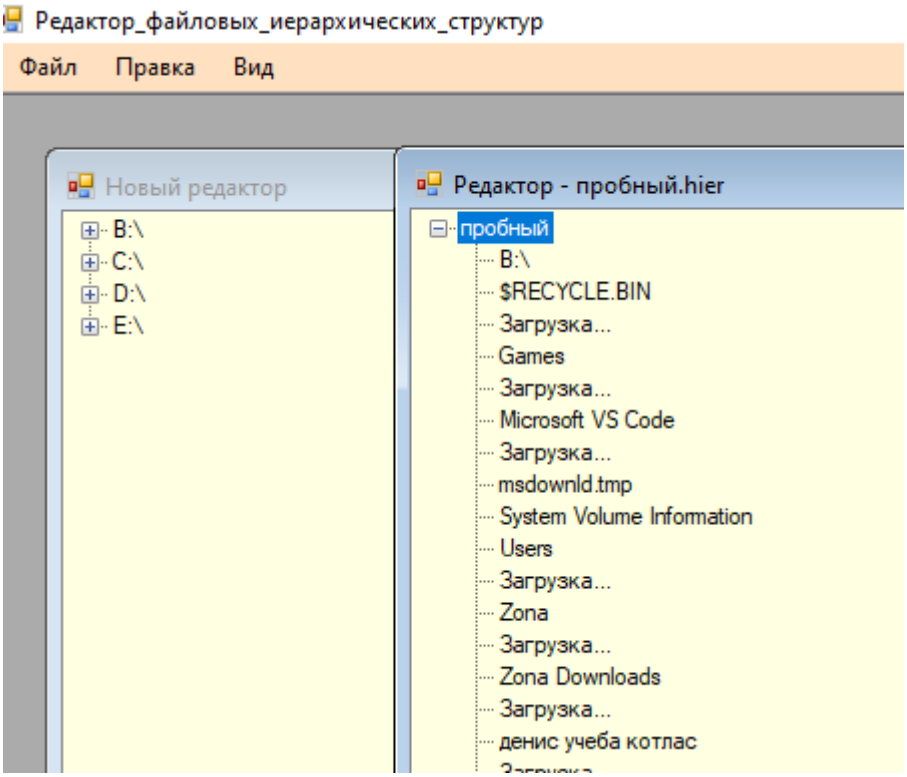

Рисунок 8. Несколько дочерних окон

5 Код созданных программных модулей

1. Главное окно Form1

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;

namespace Редактор_файловых_иерархических_структур
{
   public partial class Form1 : Form
   {
      private int documentCount = 0;

      public Form1()
      {
         InitializeComponent();
      }

      private void createMenuItem_Click(object sender, EventArgs e)
      {
         CreateNewMDIChild();
      }

      private void openMenuItem_Click(object sender, EventArgs e)
      {
         OpenFile();
      }

      private void saveMenuItem_Click(object sender, EventArgs e)
      {
         SaveFile();
      }

      private void saveAsMenuItem_Click(object sender, EventArgs e)
      {
         SaveAsToolStripMenuItemClick();
```

```
}

private void exitMenuItem_Click(object sender, EventArgs e)
{
   ExitToolStripMenuItemClick();
}

private void copyMenuItem_Click(object sender, EventArgs e)
{
   CopyToolStripMenuItemClick();
}

private void cutMenuItem_Click(object sender, EventArgs e)
{
   CutToolStripMenuItemClick();
}

private void pasteMenuItem_Click(object sender, EventArgs e)
{
   PasteToolStripMenuItemClick();
}

private void deleteMenuItem_Click(object sender, EventArgs e)
{
   DeleteToolStripMenuItemClick();
}

private void renameMenuItem_Click(object sender, EventArgs e)
{
   RenameNode();
}

private void refreshMenuItem_Click(object sender, EventArgs e)
{
   RefreshToolStripMenuItemClick();
}

private void expandAllMenuItem_Click(object sender, EventArgs e)
{
   ExpandAllToolStripMenuItemClick();
}

private void collapseAllMenuItem_Click(object sender, EventArgs e)
{
```

```
            CollapseAllToolStripMenuItemClick();
        }

        private void addMenuItem_Click(object sender, EventArgs e)
        {
            AddToolStripMenuItemClick();
        }

        private void CreateNewMDIChild()
        {
            ChildForm newMDIChild = new ChildForm();
            newMDIChild.MdiParent = this;
            newMDIChild.Text = "Новый редактор";
            newMDIChild.Show();
        }

        private void OpenFile()
        {
            OpenFileDialog openFileDialog = new OpenFileDialog();
            openFileDialog.Filter = "Файлы иерархии (*.hier)|*.hier|Все файлы
(*.*)|*.*";
            if (openFileDialog.ShowDialog() == DialogResult.OK)
            {
                if (ActiveMdiChild is ChildForm childForm)
                {
                    childForm.LoadHierarchyFromFile(openFileDialog.FileName);
                }
            }
        }

        private void SaveFile()
        {
            if (ActiveMdiChild is ChildForm childForm)
            {
                if (string.IsNullOrEmpty(childForm.CurrentFilePath))
                {
                    SaveAsFile(childForm);
                }
                else
                {
                    SaveExistingFile(childForm);
                }
            }
        }
```

```
        private void SaveAsFile(ChildForm childForm)
        {
            SaveFileDialog saveFileDialog = new SaveFileDialog();
            saveFileDialog.Filter = "Файлы иерархии (*.hier)|*.hier|Все файлы
(*.*)|*.*";
            if (saveFileDialog.ShowDialog() == DialogResult.OK)
            {
                string savedFilePath;
                childForm.SaveHierarchyToFile(saveFileDialog.FileName, out
savedFilePath);
                if (!string.IsNullOrEmpty(savedFilePath))
                {
                    childForm.CurrentFilePath = savedFilePath;
                    childForm.Text = "Редактор - " + Path.GetFileName(savedFilePath);
                }
            }
        }

        private void SaveExistingFile(ChildForm childForm)
        {
            string savedFilePath;
            childForm.SaveHierarchyToFile(childForm.CurrentFilePath, out
savedFilePath);

            if (!string.IsNullOrEmpty(savedFilePath))
            {
                childForm.Text = "Редактор - " + Path.GetFileName(savedFilePath);
            }
        }

        private void RenameNode()
        {
            // Получаем активное дочернее окно
            ChildForm activeChild = (ChildForm)this.ActiveMdiChild;
            if (activeChild != null)
            {
                activeChild.RecordTreeViewState();
                activeChild.RenameNode();
            }
            else
            {
                MessageBox.Show("Нет активного окна для переименования.",
"Переименование", MessageBoxButtons.OK, MessageBoxIcon.Warning);
```

```
        }
    }

    private void SaveAsToolStripMenuItemClick()
    {
        if (ActiveMdiChild is ChildForm childForm)
        {
            SaveFileDialog saveFileDialog = new SaveFileDialog();
            saveFileDialog.Filter = "Файлы иерархии (*.hier)|*.hier|Все файлы
(*.*)|*.*";
            if (saveFileDialog.ShowDialog() == DialogResult.OK)
            {
                string savedFilePath;
                childForm.SaveHierarchyToFile(saveFileDialog.FileName, out
savedFilePath);

                if (!string.IsNullOrEmpty(savedFilePath))
                {
                    childForm.CurrentFilePath = savedFilePath;
                    childForm.Text = "Редактор - " +
Path.GetFileName(savedFilePath);
                }
            }
        }
    }

    private void ExitToolStripMenuItemClick()
    {
        Application.Exit();
    }

    private void CopyToolStripMenuItemClick()
    {
        if (ActiveMdiChild is ChildForm childForm)
        {
            childForm.CopyNode();
        }
    }
    private void CutToolStripMenuItemClick()
    {
        if (ActiveMdiChild is ChildForm childForm)
        {
            childForm.CutNode();
        }
```

```
}

private void PasteToolStripMenuItemClick()
{
   if (ActiveMdiChild is ChildForm childForm)
   {
      childForm.RecordTreeViewState();

      childForm.PasteNode();

      childForm.RecordTreeViewState();
   }
}

private void DeleteToolStripMenuItemClick()
{
   if (ActiveMdiChild is ChildForm childForm)
   {
      childForm.RecordTreeViewState();

      childForm.DeleteNode();

      childForm.RecordTreeViewState();
   }
}
private void RefreshToolStripMenuItemClick()
{
   if (ActiveMdiChild is ChildForm childForm)
   {

      childForm.RecordTreeViewState();

      childForm.RefreshView();

      childForm.RecordTreeViewState();
   }
}

private void ExpandAllToolStripMenuItemClick()
{
   if (ActiveMdiChild is ChildForm childForm)
   {
      childForm.ExpandAllNodes();
   }
```

```
        }

        private void CollapseAllToolStripMenuItemClick()
        {
            if (ActiveMdiChild is ChildForm childForm)
            {
                childForm.CollapseAllNodes();
            }
        }

        private void AddToolStripMenuItemClick()
        {
            ChildForm activeChild = (ChildForm)this.ActiveMdiChild;

            if (activeChild != null)
            {
                string newNodeName =
Microsoft.VisualBasic.Interaction.InputBox("Введите имя нового узла:",
"Добавление узла", "Новый узел");

                if (!string.IsNullOrEmpty(newNodeName))
                {
                    // 4. Вызываем метод в ChildForm для добавления узла
                    activeChild.AddNode(newNodeName);

                    // 5. Записываем состояние дерева для undo/redo
                    activeChild.RecordTreeViewState(); // <--- ВАЖНО!
                }
            }
            else
            {
                MessageBox.Show("Нет активного окна для добавления узла.",
"Добавление узла", MessageBoxButtons.OK, MessageBoxIcon.Warning);
            }
        }
    }
}
```

2. Дочерняя форма ChildForm

```
using System;
using System.Collections.Generic;
```

```
using System.IO;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Редактор_файловых_иерархических_структур
{
    public partial class ChildForm : Form
    {
        private bool _drivesLoaded = false;
        private string _currentFilePath = null;
        private Stack<TreeViewState> undoStack = new
Stack<TreeViewState>();
        private Stack<TreeViewState> redoStack = new
Stack<TreeViewState>();
        private TreeNode _copiedNode = null;
        private bool _isCut = false;

        [Serializable]
        public class TreeViewState
        {
            public List<TreeNodeData> Nodes { get; set; }
            public string SelectedNodePath { get; set; }
        }

        [Serializable]
        public class TreeNodeData
        {
            public string Text { get; set; }
            public bool IsExpanded { get; set; }
            public List<TreeNodeData> Children { get; set; }
        }

        public string CurrentFilePath
        {
            get { return _currentFilePath; }
            set { _currentFilePath = value; }
        }

        public ChildForm()
        {
            InitializeComponent();
            this.Width = 800;
            this.Height = 600;
            this.Load += ChildForm_Load;
```

```
            UpdateUndoRedoButtons();

            renameToolStripMenuItem.Click +=
renameToolStripMenuItem_Click;
            copyToolStripMenuItem.Click += copyToolStripMenuItem_Click;
            cutToolStripMenuItem.Click += cutToolStripMenuItem_Click;
            pasteToolStripMenuItem.Click += pasteToolStripMenuItem_Click;
            deleteToolStripMenuItem.Click += deleteToolStripMenuItem_Click;
            addToolStripMenuItem.Click += addToolStripMenuItem_Click;
        }

        private void ChildForm_Load(object sender, EventArgs e)
        {
            Console.WriteLine("ChildForm_Load called");

            if (!_drivesLoaded)
            {
                LoadDrivesAsync();
                _drivesLoaded = true;
            }
        }

        private async Task LoadDrivesAsync()
        {
            DriveInfo[] drives = DriveInfo.GetDrives();

            foreach (DriveInfo drive in drives)
            {
                if (drive.IsReady) // Check if the drive is ready (e.g., not a
disconnected USB drive)
                {
                    await AddDriveNodeAsync(drive);
                }
            }
        }

        private async Task AddDriveNodeAsync(DriveInfo drive)
        {
            TreeNode driveNode = new TreeNode(drive.Name);
            driveNode.Tag = drive.Name;
            {
                bool hasDirectories = await Task.Run(() =>
Directory.GetDirectories(drive.Name).Length > 0);
```

```
                bool hasFiles = await Task.Run(() =>
Directory.GetFiles(drive.Name).Length > 0);

                if (hasDirectories || hasFiles)
                {
                    driveNode.Nodes.Add(new TreeNode("Загрузка..."));
                }
            }
            catch (UnauthorizedAccessException ex)
            {
                Console.WriteLine($"UnauthorizedAccessException while
checking drive {drive.Name}: {ex.Message}");
            }
            catch (IOException ex)
            {
                Console.WriteLine($"IOException while checking drive
{drive.Name}: {ex.Message}");
            }

            treeView1.Nodes.Add(driveNode);
        }

        private void treeView1_BeforeExpand(object sender,
TreeViewCancelEventArgs e)
        {
            TreeNode node = e.Node;

            if (node.Nodes.Count > 0 && node.Nodes[0].Text == "Загрузка...")
            {
                node.Nodes.Clear();
                LoadChildNodesAsync(node);
            }
        }

        public void LoadHierarchyFromFile(string filePath)
        {
            try
            {
                treeView1.Nodes.Clear();
                TreeNode rootNode = new
TreeNode(Path.GetFileNameWithoutExtension(filePath));
                rootNode.Tag = filePath;

                LoadDataToRootNode(rootNode, filePath);
```

```
                treeView1.Nodes.Add(rootNode);
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Ошибка при загрузке иерархии из файла:
{ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        private void LoadDataToRootNode(TreeNode rootNode, string
filePath)
        {
            try
            {
                string[] lines = File.ReadAllLines(filePath);
                foreach (string line in lines)
                {
                    rootNode.Nodes.Add(line);
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Ошибка при чтении данных из файла:
{ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        private async void LoadChildNodesAsync(TreeNode node)
        {
            string path = node.Tag.ToString();

            try
            {
                TreeNode tempNode = await Task.Run(() =>
CreatePopulatedNode(path));

                if (!IsDisposed)
                {
                    UpdateTreeNode(node, tempNode);
                }
                Console.WriteLine($"Finished loading child nodes for {path}");
            }
            catch (Exception ex)
```

```
            {
                Console.WriteLine($"Error loading child nodes for {path}:
{ex.Message}");
            }
        }

        private TreeNode CreatePopulatedNode(string path)
        {
            TreeNode tempNode = new TreeNode(Path.GetFileName(path));
            tempNode.Tag = path;

            PopulateNodes(tempNode, path, 3);

            return tempNode;
        }

        private void UpdateTreeNode(TreeNode node, TreeNode tempNode)
        {
            if (treeView1.InvokeRequired)
            {
                treeView1.Invoke(new Action(() =>
                {
                    node.Nodes.Clear();
                    foreach (TreeNode child in tempNode.Nodes)
                    {
                        node.Nodes.Add(child);
                    }
                }));
            }
            else
            {
                node.Nodes.Clear();
                foreach (TreeNode child in tempNode.Nodes)
                {
                    node.Nodes.Add(child);
                }
            }
        }

        public void SaveHierarchyToFile(string filePath, out string
savedFilePath)
        {
            try
            {
```

```
                List<string> lines = new List<string>();
                foreach (TreeNode node in treeView1.Nodes)
                {
                    CollectNodeText(node, lines);
                }
                File.WriteAllLines(filePath, lines);

                MessageBox.Show("Иерархия успешно сохранена в файл.",
"Сохранение", MessageBoxButtons.OK, MessageBoxIcon.Information);

                savedFilePath = filePath;
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Ошибка при сохранении иерархии в
файл: {ex.Message}", "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                savedFilePath = null;
            }
        }

        public void CopyNode()
        {
            if (treeView1.SelectedNode != null)
            {
                CopySelectedNode();
            }
            else
            {
                MessageBox.Show("Пожалуйста, выберите узел для
копирования.", "Копирование", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
            }
        }

        private void CopySelectedNode()
        {
            _copiedNode = (TreeNode)treeView1.SelectedNode.Clone();
            _isCut = false;
            MessageBox.Show("Узел скопирован.", "Копирование",
MessageBoxButtons.OK, MessageBoxIcon.Information);
        }

        public void CutNode()
```

```
        {
            if (treeView1.SelectedNode != null)
            {
                CutSelectedNode();
            }
            else
            {
                MessageBox.Show("Пожалуйста, выберите узел для
вырезания.", "Вырезание", MessageBoxButtons.OK, MessageBoxIcon.Warning);
            }
        }

        private void CutSelectedNode()
        {
            _copiedNode = (TreeNode)treeView1.SelectedNode.Clone();
            _isCut = true;
            MessageBox.Show("Узел вырезан.", "Вырезание",
MessageBoxButtons.OK, MessageBoxIcon.Information);
        }

        public void PasteNode()
        {
            if (_copiedNode != null && treeView1.SelectedNode != null)
            {
                PasteCopiedNode();
            }
            else
            {
                MessageBox.Show("Пожалуйста, сначала скопируйте или
вырежьте узел, а затем выберите родительский узел для вставки.", "Вставка",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
            }
        }

        private void PasteCopiedNode()
        {
            TreeNode newNode = (TreeNode)_copiedNode.Clone();
            if (_isCut)
            {
                RemoveCutNode();
            }
            treeView1.SelectedNode.Nodes.Add(newNode);
            treeView1.SelectedNode.Expand();
```

```
            MessageBox.Show("Узел вставлен.", "Вставка",
MessageBoxButtons.OK, MessageBoxIcon.Information);
        }

        private void RemoveCutNode()
        {
            TreeNode nodeToRemove = treeView1.SelectedNode;
            if (nodeToRemove.Parent != null)
            {
                nodeToRemove.Parent.Nodes.Remove(nodeToRemove);
            }
            else
            {
                treeView1.Nodes.Remove(nodeToRemove);
            }
            _isCut = false;
            _copiedNode = null;
        }

        public void DeleteNode()
        {
            if (treeView1.SelectedNode != null)
            {
                DeleteSelectedNode();
            }
            else
            {
                MessageBox.Show("Пожалуйста, выберите узел для
удаления.", "Удаление", MessageBoxButtons.OK, MessageBoxIcon.Warning);
            }
        }

        private void DeleteSelectedNode()
        {
            TreeNode nodeToRemove = treeView1.SelectedNode;

            if (nodeToRemove.Parent != null)
            {
                nodeToRemove.Parent.Nodes.Remove(nodeToRemove);
            }
            else
            {
                treeView1.Nodes.Remove(nodeToRemove);
            }
```

```
                MessageBox.Show("Узел удален.", "Удаление",
MessageBoxButtons.OK, MessageBoxIcon.Information);
            }

            public void RenameNode()
            {
                if (treeView1.SelectedNode != null)
                {
                    StartNodeRenaming();
                }
                else
                {
                    MessageBox.Show("Пожалуйста, выберите узел для
переименования.", "Переименование", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
                }
            }

            private void StartNodeRenaming()
            {
                treeView1.LabelEdit = true;
            }

            private void treeView1_AfterLabelEdit(object sender,
NodeLabelEditEventArgs e)
            {
                HandleAfterLabelEdit(e);
            }

            private void HandleAfterLabelEdit(NodeLabelEditEventArgs e)
            {
                if (e.Label != null)
                {

                    if (e.Label.Trim().Length > 0)
                    {
                        e.Node.Text = e.Label.Trim();
                        e.Node.EndEdit(false);
                    }
                    else
                    {
                        CancelNodeEdit(e, "Нельзя задать пустое имя узла.");
                    }
```

```
            }
            else
            {
                CancelNodeEdit(e, null);
            }
        }

        private void CancelNodeEdit(NodeLabelEditEventArgs e, string
message)
        {
            e.CancelEdit = true;
            e.Node.EndEdit(true);
            if (!string.IsNullOrEmpty(message))
            {
                MessageBox.Show(message, "Переименование",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
            }
        }

        public void RefreshView()
        {
            treeView1.Nodes.Clear();
            _drivesLoaded = false;
            ChildForm_Load(this, EventArgs.Empty);
            MessageBox.Show("Дерево каталогов обновлено.",
"Обновление", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }

        public void ExpandAllNodes()
        {
            treeView1.ExpandAll();
            MessageBox.Show("Все узлы развернуты.", "Развернуть все",
MessageBoxButtons.OK, MessageBoxIcon.Information);
        }

        public void CollapseAllNodes()
        {
            treeView1.CollapseAll();
            MessageBox.Show("Все узлы свернуты.", "Свернуть все",
MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        public void AddNode(string newNodeName)
        {
            AddNodeToTreeView(newNodeName);
```

```
        }

        private void AddNodeToTreeView(string newNodeName)
        {
            TreeNode selectedNode = treeView1.SelectedNode;
            if (selectedNode != null)
            {
                string selectedPath = GetFullPath(selectedNode);

                string newFolderPath = Path.Combine(selectedPath,
newNodeName);

                try
                {
                    Directory.CreateDirectory(newFolderPath);

                    TreeNode newNode = new TreeNode(newNodeName);
                    selectedNode.Nodes.Add(newNode);
                    selectedNode.Expand();

                    treeView1.SelectedNode = newNode;
                    treeView1.LabelEdit = true;
                    newNode.BeginEdit();
                }
                catch (Exception ex)
                {
                    MessageBox.Show("Не удалось создать папку: " +
ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            }
            else
            {
                MessageBox.Show("Выберите родительский узел для
добавления.", "Добавление узла", MessageBoxButtons.OK,
MessageBoxIcon.Information);
            }
        }
        private string GetFullPath(TreeNode node)
        {
            string path = node.Text;
            TreeNode parent = node.Parent;
            while (parent != null)
            {
                path = Path.Combine(parent.Text, path);
```

```
            parent = parent.Parent;
        }
        return path;
    }

    private string GetNodePath(TreeNode node)
    {
        string path = node.Text;
        TreeNode parent = node.Parent;
        while (parent != null)
        {
            path = parent.Text + "/" + path;
            parent = parent.Parent;
        }
        return path;
    }

    private TreeNode FindNodeByPath(TreeNodeCollection nodes, string
path)
    {
        string[] parts = path.Split('/');
        TreeNode currentNode = null;

        TreeNodeCollection currentNodes = nodes;
        for (int i = 0; i < parts.Length; i++)
        {
            bool found = false;
            foreach (TreeNode node in currentNodes)
            {
                if (node.Text == parts[i])
                {
                    currentNode = node;
                    currentNodes = node.Nodes;
                    found = true;
                    break;
                }
            }
            if (!found) return null;
        }

        return currentNode;
    }

    public TreeViewState CaptureTreeViewState()
```

```
        {
            TreeViewState state = new TreeViewState();
            state.Nodes = CaptureNodes(treeView1.Nodes);

            // Сохраняем путь к выбранному узлу
            if (treeView1.SelectedNode != null)
            {
                state.SelectedNodePath = GetNodePath(treeView1.SelectedNode);
            }

            return state;
        }

        private List<TreeNodeData> CaptureNodes(TreeNodeCollection
nodes)
        {
            List<TreeNodeData> dataList = new List<TreeNodeData>();
            foreach (TreeNode node in nodes)
            {
                TreeNodeData data = new TreeNodeData
                {
                    Text = node.Text,
                    IsExpanded = node.IsExpanded,
                    Children = CaptureNodes(node.Nodes)
                };
                dataList.Add(data);
            }
            return dataList;
        }

        public void RecordTreeViewState()
        {
            BeforeTreeViewChange();
        }

        private void BeforeTreeViewChange()
        {
            redoStack.Clear();
            undoStack.Push(CaptureTreeViewState());

            UpdateUndoRedoButtons();
        }
```

```
        public void RestoreTreeViewState(TreeView treeView, TreeViewState
state)
        {
            treeView.Nodes.Clear();
            RestoreNodes(treeView.Nodes, state.Nodes);

            if (!string.IsNullOrEmpty(state.SelectedNodePath))
            {
                TreeNode selectedNode = FindNodeByPath(treeView.Nodes,
state.SelectedNodePath);
                if (selectedNode != null)
                {
                    treeView.SelectedNode = selectedNode;
                }
            }
        }

        private void RestoreNodes(TreeNodeCollection nodes,
List<TreeNodeData> dataList)
        {
            foreach (TreeNodeData data in dataList)
            {
                TreeNode node = new TreeNode(data.Text);
                nodes.Add(node);
                RestoreNodes(node.Nodes, data.Children);

                if (data.IsExpanded)
                {
                    node.Expand();
                }
            }
        }

        private void CollectNodeText(TreeNode node, List<string> lines)
        {
            lines.Add(node.Text);

            foreach (TreeNode childNode in node.Nodes)
            {
                CollectNodeText(childNode, lines);
        }

        private void PopulateNodes(TreeNode parentNode, string path, int
depth)
```

```
        {
            if (depth <= 0)
            {
                Console.WriteLine("PopulateNodes: depth limit reached,
returning");
                return;
            }

            Console.WriteLine("PopulateNodes: path = " + path + ", depth = " +
depth);

            try
            {
                PopulateDirectories(parentNode, path, depth);
                PopulateFiles(parentNode, path);
            }
            catch (UnauthorizedAccessException ex)
            {
                Console.WriteLine("UnauthorizedAccessException in
PopulateNodes: " + ex.Message + ", path = " + path);
            }
            catch (IOException ex)
            {
                Console.WriteLine("IOException in PopulateNodes: " +
ex.Message + ", path = " + path);
            }
            catch (Exception ex)
            {
                Console.WriteLine("Exception in PopulateNodes: " + ex.Message
+ ", path = " + path);
            }
        }

        private void PopulateDirectories(TreeNode parentNode, string path, int
depth)
        {
            string[] directories = Directory.GetDirectories(path);
            foreach (string directory in directories)
            {
                TreeNode node = CreateDirectoryNode(directory);
                parentNode.Nodes.Add(node);
            }
        }
```

```
        private TreeNode CreateDirectoryNode(string directory)
        {
            TreeNode node = new TreeNode(Path.GetFileName(directory));
            node.Tag = directory;
            {
                if (Directory.GetDirectories(directory).Length > 0 ||
Directory.GetFiles(directory).Length > 0)
                {
                    node.Nodes.Add(new TreeNode("Загрузка..."));
                }
            }
            catch (UnauthorizedAccessException ex)
            {
                Console.WriteLine($"UnauthorizedAccessException while
checking directory {directory}: {ex.Message}");
            }
            catch (IOException ex)
            {
                Console.WriteLine($"IOException while checking directory
{directory}: {ex.Message}");
            }
            return node;
        }

        private void PopulateFiles(TreeNode parentNode, string path)
        {
            string[] files = Directory.GetFiles(path);
            foreach (string file in files)
            {
                TreeNode node = new TreeNode(Path.GetFileName(file));
                node.Tag = file;
                parentNode.Nodes.Add(node);
            }
        }
        private void Undo()
        {
            if (undoStack.Count > 0)
            {
                PerformUndo();
            }
        }

        private void PerformUndo()
        {
```

```
            redoStack.Push(CaptureTreeViewState());
            TreeViewState previousState = undoStack.Pop();
            RestoreTreeViewState(treeView1, previousState);

            UpdateUndoRedoButtons();
        }

        private void Redo()
        {
            if (redoStack.Count > 0)
            {
                PerformRedo();
            }
        }

        private void PerformRedo()
        {
            undoStack.Push(CaptureTreeViewState());
            TreeViewState nextState = redoStack.Pop();
            RestoreTreeViewState(treeView1, nextState);

            UpdateUndoRedoButtons();
        }

        private void UpdateUndoRedoButtons()
        {
            buttonUndo.Enabled = undoStack.Count > 0;
            buttonRedo.Enabled = redoStack.Count > 0;
        }

        private void buttonUndo_Click(object sender, EventArgs e)
        {
            Undo();
        }

        private void buttonRedo_Click(object sender, EventArgs e)
        {
            Redo();
        }

        private void renameToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            RecordTreeViewState();
```

```
            RenameNode();
            RecordTreeViewState();
        }

        private void copyToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            CopyNode();
        }

        private void cutToolStripMenuItem_Click(object sender, EventArgs e)
        {
            CutNode();
        }

        private void pasteToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            RecordTreeViewState();

            PasteNode();

            RecordTreeViewState();
        }

        private void deleteToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            RecordTreeViewState();

            DeleteNode();

            RecordTreeViewState();
        }

        private void addToolStripMenuItem_Click(object sender, EventArgs e)
        {
            string newNodeName =
Microsoft.VisualBasic.Interaction.InputBox("Введите имя нового узла:",
"Добавление узла", "Новый узел");

            if (!string.IsNullOrEmpty(newNodeName))
            {
                RecordTreeViewState();
```

```
                AddNode(newNodeName);

                RecordTreeViewState();
            }
        }
        private void treeView1_NodeMouseClick(object sender,
TreeNodeMouseClickEventArgs e)
        {
            if (e.Button == MouseButtons.Right)
            {
                treeView1.SelectedNode = e.Node;
            }
        }
    }
}
```

6 Заключение

В рамках данного курсового проекта была разработана программа «Редактор файловых иерархических структур», предназначенная для визуализации, редактирования и управления файловой системой в удобном графическом интерфейсе. Проект успешно реализован и позволяет пользователю выполнять основные операции с файлами и каталогами, такие как просмотр, создание, переименование, копирование, вырезание, вставка и удаление элементов файловой системы.

7 Список использованной литературы:

1. Джон Скит. C# для профессионалов: тонкости программирования, 3-е издание, новый перевод = C# in Depth, 3rd ed.. — М.: «Вильямс», 2014.
2. Джозеф Албахари, Бен Албахари. C# 6.0. Справочник. Полное описание языка = C# 6.0 in a Nutshell: The Definitive Reference. — М.: «Вильямс», 2018.
3. Кристиан Нейгел, Карли Уотсон и др. Visual C# 2010: полный курс = Beginning Microsoft Visual C# 2010. — М.: Диалектика, 2010.