


Лабораторная работа «Создание деревьев, списков, полос прокруток»

Цель: Овладение навыками создания и практического использования элементов управления дерева, списки, списки изображений и полосы прокруток.

1. Теоретическая часть.

1.1. Элементы управления «Список», «Дерево» и «Список изображений».

Список (ListView)  предназначен для отображения списков данных. Компонент ListView отличается от ListBox расширенным списком возможностей: установка пиктограмм для каждого элемента списка, отображение данных в нескольких колонках, несколько стилей представления элементов списка. Типичным примером использования элемента ListView является правая сторона программы «Проводник», которая входит в стандартную поставку Microsoft Windows (рис. 3.1). В клиентской области программы «Проводник» находятся два элемента управления. Элемент управления «дерево» (tree-view) слева — это иерархический список дисководов и каталогов пользователя. Элемент управления «список» (list-view) справа показывает подкаталоги и файлы выбранного каталога в одном из четырех форматов: простой список, таблица из нескольких столбцов, имена файлов с мелкими значками или имена с крупными значками.

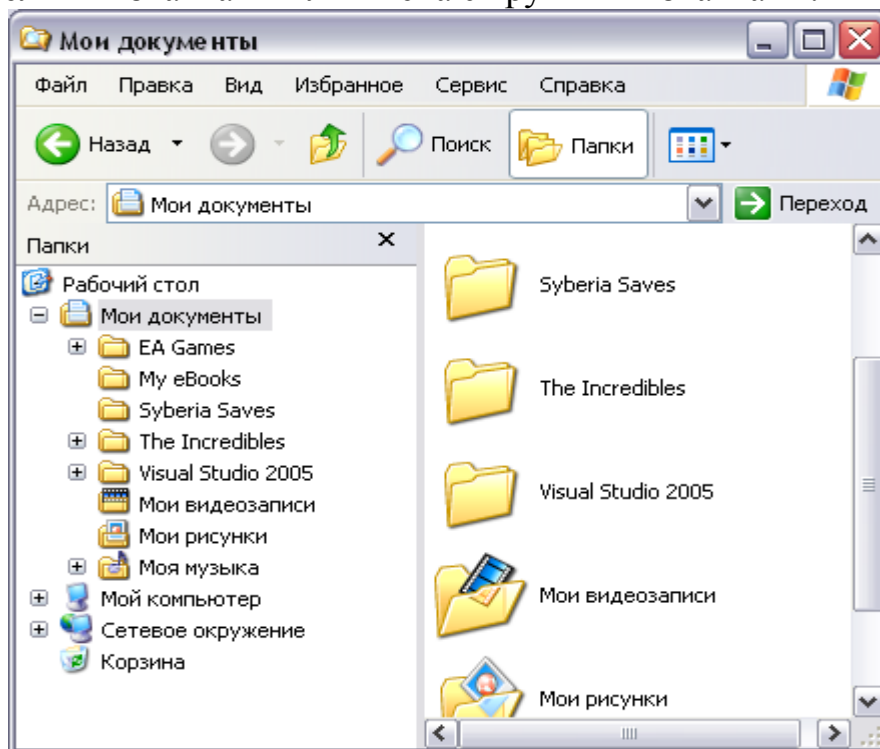




Рис. 3.1. Проводник ОС Windows.

Дерево (TreeView)  предназначено для отображения данных в виде дерева. Элементы представления начинаются с корня дерева и отображаются вглубь. Примером может служить левая сторона программы «Проводник», которая отображает дерево каталогов (рис. 3.1).

В левой половине окна располагается содержимое всего диска компьютера. Все элементы списка имеют общий корень «Рабочий стол», то есть все элементы являются подпунктами одного общего корня. В правой стороне отображается

содержимое текущей выделенной папки. В данном случае это папка «Мои документы». Правая панель отображает элементы в виде списка с пиктограммами.

Список изображений (ImageList)  ImageList относится к невидимым вспомогательным элементам управления. ImageList хранит изображения, отображаемые другими элементами управления. ImageList используется такими компонентами, как ListView, TreeView, Button и т. д. Элемент ImageList содержит список изображений. Если какому-либо объекту необходимо отобразить изображение, то необходимо задать для соответствующего свойства лишь индекс элемента в списке изображений. ImageList может содержать изображения в формате BMP, JPG, GIF, PNG, ICO, WMF, EMF. Для того чтобы построить список изображений, в Visual Studio 2005 разработан мастер Image Collection Editor.

1.2. Полосы прокруток.

Горизонтальные и вертикальные полосы прокрутки широко используются в приложениях Windows. Они обеспечивают интуитивный способ передвижения по спискам информации и позволяют делать поля для ввода данных очень большими. Полоса прокрутки состоит из трех областей, которые нажимаются или перемещаются для изменения значения полосы прокрутки.

Рассмотрим работу элемента на примере горизонтальной полосы прокрутки. Нажатие левой стрелки уменьшает значение положения бегунка на минимальное. Нажатие правой стрелки увеличивает значение положения бегунка на минимальное. При нажатии курсором мыши в области полосы прокрутки значение положения бегунка изменяется на величину, большую, чем значение при нажатии на кнопки «влево» и «вправо». При использовании реквизитов полос прокрутки мы можем полностью определять, как работает каждый из них. Позиция бегунка — единственная выходная информация из полосы прокрутки.

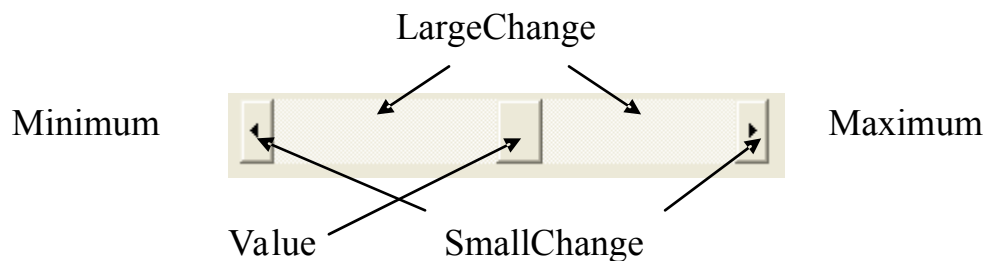


Рис. 3.2. Свойства полос прокрутки.

LargeChange — значение, которое добавляется или вычитается из значения текущего положения бегунка; это величина, на которую изменяется положение бегунка при нажатии курсора в области полосы прокрутки.

Maximum — значение горизонтальной полосы прокрутки в крайнем правом положении и значение вертикальной полосы прокрутки в крайнем нижнем положении. Может принимать значения от -32 768 до 32 767.

Minimum — другое предельное значение — для горизонтальной полосы прокрутки в крайнем левом положении и для вертикальной полосы прокрутки в крайнем верхнем. Может принимать значения от -32 768 до 32 767.

SmallChange — значение, которое добавляется или вычитается из значения текущего положения бегунка; значение, на которое изменяется положение полосы

прокрутки, когда нажата любая из стрелок прокрутки.

Value — текущая позиция бегунка внутри полосы прокрутки. При изменении значения свойства Value бегунок перемещается в соответствующую позицию.

События полосы прокрутки:

Change — событие, генерируемое после того, как позиция бегунка изменилась. Это событие используется для получения нового значения, чтобы считать новое значение положения бегунка после любых действий на полосе прокрутки.

Scroll — событие, вызываемое непрерывно всякий раз, когда бегунок передвигается.

2. Практическая часть.

2.1. Создание форм с элементами управления «Список» и «Дерево».

Необходимо создать приложение, которое будет способно добавлять и удалять элементы в дерево и в список. Главное окно программы будет содержать поле ввода данных, две кнопки: *добавить в список* и *Добавить в дерево*, компонент ListView и компонент TreeView. В поле ввода пользователь может набрать любую строку. При нажатии на соответствующую кнопку содержимое поля переносится в ListView или TreeView.

Для этого создайте новое C# Windows-приложение под названием ViewApp. Добавьте на форму следующие элементы:

- ListView;
- TreeView;
- Button—два элемента;
- TextBox.

Примерное расположение элементов показано на рис. 3.3. Вы можете построить интерфейс программы по своему усмотрению.

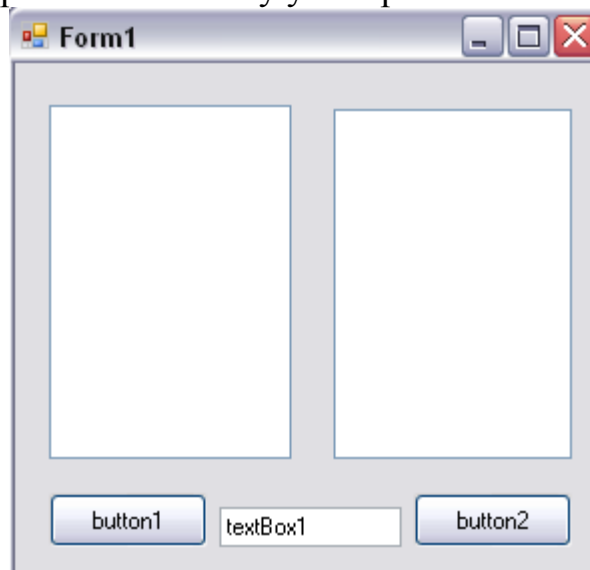


Рис. 3.3. Проектирование формы приложения ViewApp.

Измените свойства добавленных компонент:

button1:

Text — «добавить в список»

button2:

Text — «добавить в дерево»

textBox1:

Text — «»

Теперь необходимо наделить код функциональностью. В первую очередь следует добавить обработчик нажатия кнопки *Добавить в список*. Для этого щелкните два раза указателем мыши по кнопке на форме. Перед вами откроется окно кода, в который будет добавлена функция `button1_Click`. Добавьте в функцию `button1_Click` код, представленный ниже.

```
private void button1_Click(object sender,
System.EventArgs e)
{    // добавляем новый элемент в список
    listView1.Items.Add( textBox1.Text);
}
```

Откомпилируйте и запустите программу. Выполните описанную ниже последовательность действий: наберите в поле ввода «Пять». Нажмите кнопку *Добавить в список*. Наберите в поле ввода «Три». Нажмите кнопку *Добавить в список*. Наберите в поле ввода «Один». Нажмите кнопку *Добавить в список*. Элементы расположатся в списке последовательно. Каждый новый элемент будет добавлен в конец списка.

Вторым объектом создаваемой программы является дерево. Давайте рассмотрим работу с деревом. Для начала необходимо добавить обработчик кнопки *Добавить в дерево*. Для этого щелкните два раза левой кнопкой мыши по кнопке на форме. При этом в программу добавится обработчик нажатия кнопки `button2` — функция `button2_Click`. Измените код функции `button2_Click` так, как представлено ниже:

```
private void button2_Click(object sender,
System.EventArgs e)
{    // получаем выделенный узел
    TreeNode node = treeView1.SelectedNode;
    // если выделенного узла нет
    if (node == null)
    {    // добавляем новый элемент
        // в корень основного дерева
        treeView1.Nodes.Add(textBox1.Text) ;
    }
    // если имеется выделенный узел
    else
    {    // добавляем новый элемент
        // как вложенный в выделенный узел
        node.Nodes.Add(textBox1.Text);}
}
```

Откомпилируйте и запустите программу. Выполните описанную ниже последовательность действий. При этом не делайте выделения элементов дерева, пока об этом не будет сказано в описании.

Наберите в поле ввода строку «1-узел», нажмите кнопку *Добавить в дерево*.

Наберите в поле ввода строку «2-узел», нажмите кнопку *Добавить в дерево*.

Наберите в поле ввода строку «3-узел», нажмите кнопку *Добавить в дерево*.

Выделите элемент «1-узел». Наберите в поле ввода строку «1-1-узел», нажмите кнопку *Добавить в дерево*.

Выделите элемент «1-1-узел». Наберите в поле ввода строку «1-1-1-узел», нажмите кнопку *Добавить в дерево*.

Выделите элемент «1-1-узел». Наберите в поле ввода строку «1-1-2-узел», нажмите кнопку *Добавить в дерево*.

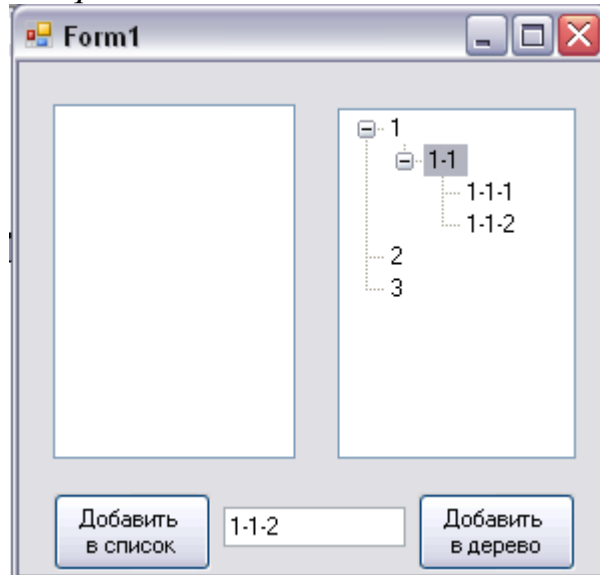


Рис. 3.4. Результаты работы с деревом.

Дерево дает более широкий спектр представления информации. Элементы дерева могут быть встроены в другие элементы. Каждый элемент дерева называется узлом. Если узел содержит другие узлы, то называется корнем дерева. В нашем случае элементы «1-узел» и «1-1-узел» являются корнями для других элементов. Каждый элемент дерева может стать корнем, если встроить в него другие элементы дерева. Корень может содержать сколько угодно элементов одного уровня. В нашем случае узел «1-1-узел» содержит два элемента одного уровня «1-1-1-узел» и «1-1-2-узел».

Давайте рассмотрим код, который заставляет программу работать именно таким образом:

```
TreeNode node = treeView1.SelectedNode;
```

Класс `TreeView` имеет свойство `SelectedNode`. Это свойство возвращает объект `TreeNode`, который выделен на текущий момент в дереве. Если в дереве не выделен ни один элемент, то свойство `SelectedNode` возвращает `null`.

```
if (node == null)
{
    treeView1.Nodes.Add(textBox1.Text) ;
}
```

Если все-таки свойство `SelectedNode` вернет `null`, то элемент будет добавлен в список первого уровня. Так были добавлены «1-узел», «2-узел» и «3-узел». Свойство `Nodes` класса `TreeView` содержит коллекцию `TreeNodeCollection` всех узлов первого уровня. Функция `Add` класса `TreeNodeCollection` добавляет новый элемент в конец коллекции.

```
else
```

```

{
    node.Nodes.Add(textBox1.Text);
}

```

Если же один из элементов дерева выделен, то переменная `node` будет содержать выделенный объект. Класс `TreeNode` имеет свойство `Nodes`, которое так же, как и свойство `Nodes` объекта `TreeView`, возвращает объект класса `TreeNodeCollection`. Каждый объект `TreeNode` содержит коллекцию узлов, для которых он является родительским. Поэтому, когда вы выделили узел «1-узел», новый элемент был добавлен в коллекцию его дочерних узлов. Как и при выделении элемента «1-1-элемент», в коллекцию его дочерних узлов были добавлены элементы «1-1-1-элемент» и «1-1-2-элемент».

2.2. Использование списка изображений в списке.

Откройте приложение `ViewApp`. Добавьте на форму компонент `ImageList`. Он отобразится не на форме, а в панели компонент ниже формы. Компонент `ImageList` имеет свойство `Images` (рис. 3.5).

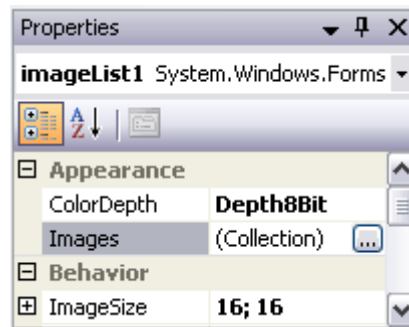


Рис. 3.5. Окно свойств `ImageList`.

Это свойство отвечает за коллекцию изображений, содержащихся в списке. Откройте окно редактирования `Image Collection Editor`, нажав кнопку с тремя точками в поле `Collection` окна свойств. Перед вами откроется окно `Image Collection Editor` (рис. 3.6).

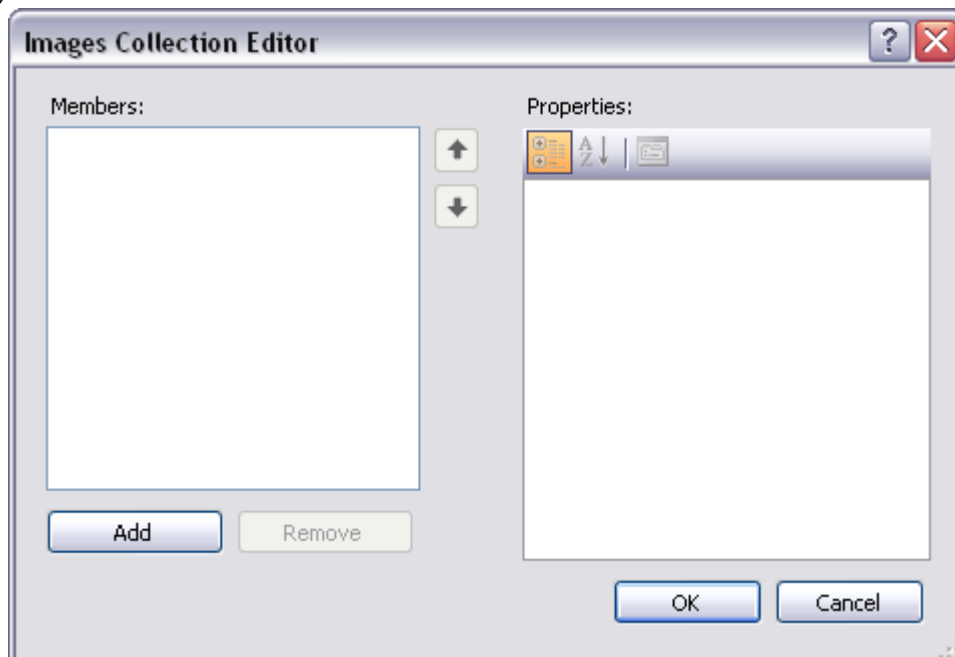


Рис. 3.6. Окно настройки коллекции изображений.

Это окно позволяет добавить новое изображение в коллекцию или удалить

уже существующее. Для того чтобы добавить новое изображение, вам необходимо нажать кнопку *Add*. При этом Visual Studio 2005 предложит вам выбрать файл, содержащий изображение в одном из доступных форматов. Вы можете выбрать любой файл, по своему усмотрению. Например, файл «wc.gif» (или любой другой). При этом миниатюрное изображение пиктограммы появится в списке Members окна (рис. 3.7).

Список Members может содержать сколько угодно элементов. Для нашего примера будет достаточно одного элемента в списке. Закройте Image Collection Editor, нажав кнопку *OK*.

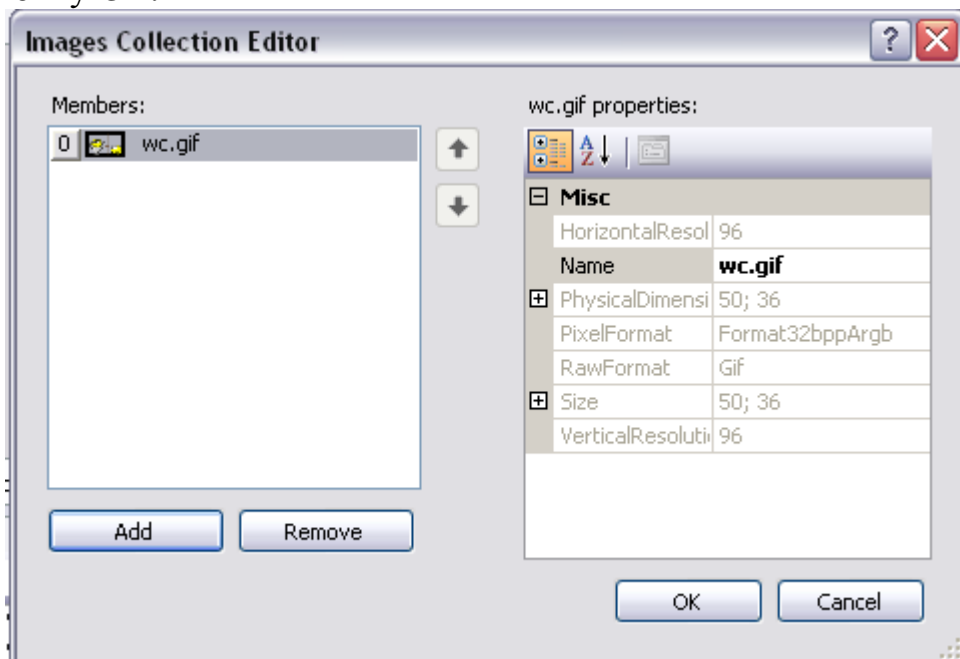


Рис. 3.7. Добавление нового изображения в список.

Компонент `ListView` имеет свойство `LargeImageList`, которое хранит список изображений, закрепленный за `ListView` объектом. Установите в окне свойств объекта `listView1` свойство `LargeImageList` как `imageList1`. Теперь за списком `listView1` будет закреплен список изображений `imageList1`.

Далее давайте изменим код нашей программы, для того чтобы список мог содержать не только текстовую информацию, но и графическую. Программа потребует минимальных изменений. Класс `ListViewItemsCollection` содержит несколько вариантов функции `Add`. Мы использовали `Add(string text)`. Другим вариантом функции является `Add(string text, int imageIndex)`. Вторым параметром этой функции является индекс изображения в списке `ImageList`, которое было предварительно установлено для `ListView`.

Замените вызов функции

```
listView1.Items.Add( textBox1.Text );
```

внутри функции `button1_Click` на

```
listView1.Items.Add( textBox1.Text, 0 );
```

Добавился лишь второй параметр при вызове функции. Однако попробуйте вновь запустить программу. Добавьте в список элементы «один» и «два». На этот раз список будет содержать и текстовое представление элемента, и его пиктограмму (рис. 3.7).

Для всех элементов нашего списка пиктограмма будет одна и та же. Однако вы можете установить в качестве пиктограммы любой элемент списка изображений.

Вторым параметром функции `Add` является индекс изображения из списка `ImageList`. Если `ImageList` содержит пять элементов, то для установки пиктограммы с индексом 3 вам необходимо вызвать функцию `Add` как:

```
listView1.Items.Add( textBox1.Text, 2 );
```

2.3. Использование списка изображений в дереве.

Так же, как пиктограммы в `ListView` компоненте, пиктограммы могут использоваться другими компонентами. Использование `ImageList` схоже в применении со всеми компонентами. Вот как выглядят `ImageList` вместе с `TreeView`. В окне свойств объекта `treeView1` установите свойство `ImageList` в `imageList1`.

Для того чтобы добавить отображение пиктограммы в дерево, измените код функции `button2_Click` на следующий код:

```
// получаем выделенный узел
TreeNode node = treeView1.SelectedNode;
// если выделенного узла нет
if (node == null)
{
    // добавляем новый элемент
    // в корень основного дерева
    TreeNode newNode = new TreeNode();
    newNode.Text = textBox1.Text;
    newNode.ImageIndex = 0;
    treeView1.Nodes.Add(newNode);
}
// если имеется выделенный узел
else
{
    // добавляем новый элемент
    // как вложенный в выделенный узел
    TreeNode newNode = new TreeNode();
    newNode.Text = textBox1.Text;
    newNode.ImageIndex = 0;
    node.Nodes.Add(newNode);
}
```

Функция `Add` класса `TreeNodeCollection` так же перегружена, как и функция `Add` класса `ListViewItemsCollection`. Вторым вариантом функции `Add` является `Add(System.Windows.Forms.TreeNode node)`. Для того чтобы добавить текстовый элемент с пиктографическим изображением в дерево, необходимо:

- установить у `TreeView` свойство `ImageList`;
- создать элемент `TreeNode`;
- установить свойство `Text`;
- установить свойство `ImageIndex`;
- вызвать функцию `TreeNode.Nodes.Add()`.

Запустите программу с новым кодом. Попробуйте добавить элементы в дерево. Так же, как и в списке, все узлы дерева будут отображаться с пиктограммой (рис. 3.8.).

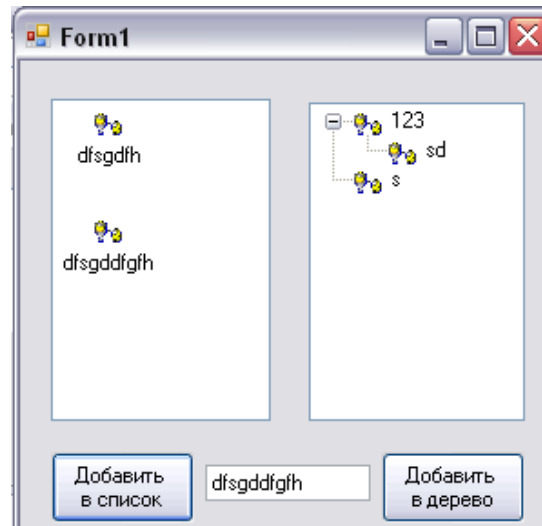


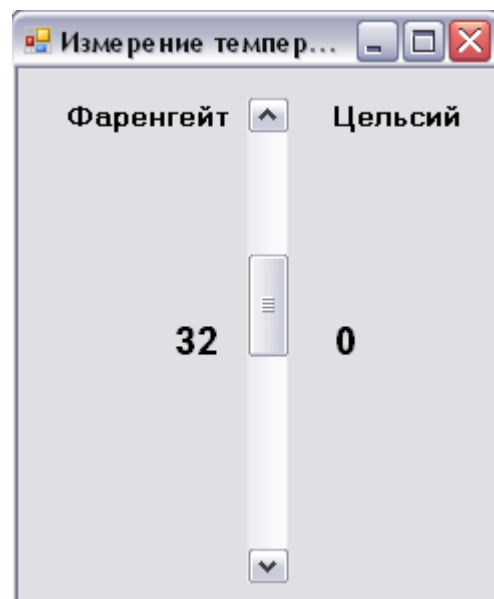
Рис. 3.8. Использование пиктограмм компонентом TreeView.

2.4. Создание полос прокруток на форме.

Создайте новый Windows Forms проект с именем ScrollApp. Поместите на форму элемент VScrollBar и четыре элемента Label. Сделайте это так, как показано на рис. 3.9, а.



а) этап 1



б) этап 2

Рис. 3.9. Проектирование формы приложения ScrollApp.

Установите для компонент свойства в соответствии с приведенным ниже списком:

Form1:

Text – Измерение температуры

vScrollBar1:

LargeChange – 10

Maximum – 60

Minimum – 20

SmallChange – 1

Value – 32

label1:

Text – Фаренгейт

```

Font Size - 10
Font Bold - True
label2:
BackColor - White
Text - 32
Font Size - 14
Font Bold - True
Name - labelFarTemp
label3:
Text - Цельсий
Font Size - 10
FontBold - True
label4:
BackColor - White
Text - 0
Font Size - 14
Font Bold - True
Name - labelCTemp

```

Обратите внимание, что значения температур проинициализированы 32 и 0. Когда вы сделаете свою форму, она должна иметь вид, представленный на рис. 3.9, б. Как уже отмечалось, компонент ScrollBar имеет событие Scroll. Давайте добавим обработчик этого события в код программы. Для этого в окне свойств на закладке событий для элемента vScrollBar1 щелкните два раза по событию Scroll. В код программы добавится обработчик события Scroll:

```

private void vScrollBar1_Scroll(object sender,
System.Windows.Forms.ScrollEventArgs e) { }

```

Добавьте в функцию vScrollBar1_Scroll код, представленный ниже:

```

private void vScrollBar1_Scroll(object sender,
System.Windows.Forms.ScrollEventArgs e)
{
    labelFarTemp.Text = vScrollBar1.Value.ToString() ;
    labelCTemp.Text = Convert.ToString((int) ((
(double)vScrollBar1.Value - 32)/9*5));
}

```

Этот код переводит значение Value полосы прокрутки во время изменения положения бегунка и определяет значение температуры по шкале Фаренгейта. Затем вычисляется значение температуры по Цельсию и отображаются оба значения. Откомпилируйте и запустите программу.

Вы увидите, что при изменении положения бегунка полосы прокрутки изменяются значения температур. Попробуйте найти точку, в которой значение температуры по Цельсию равно значению температуры по Фаренгейту.

3. Задания.

1. Выполните все примеры, приведенные в практической части лабораторной работы.
2. Измените первый пример следующим образом. Добавьте на форму, содержащую элементы Список и Дерево, кнопки для удаления выделенного (или выделенных) элементов из списка и дерева.

3. Создайте приложение, показанное на рис. 3.10. Пользователю должна быть предоставлена возможность выбирать элементы в `CheckedListBox`, а затем посредством нажатия кнопки переносить выбранные элементы в обычный список `ListBox`.

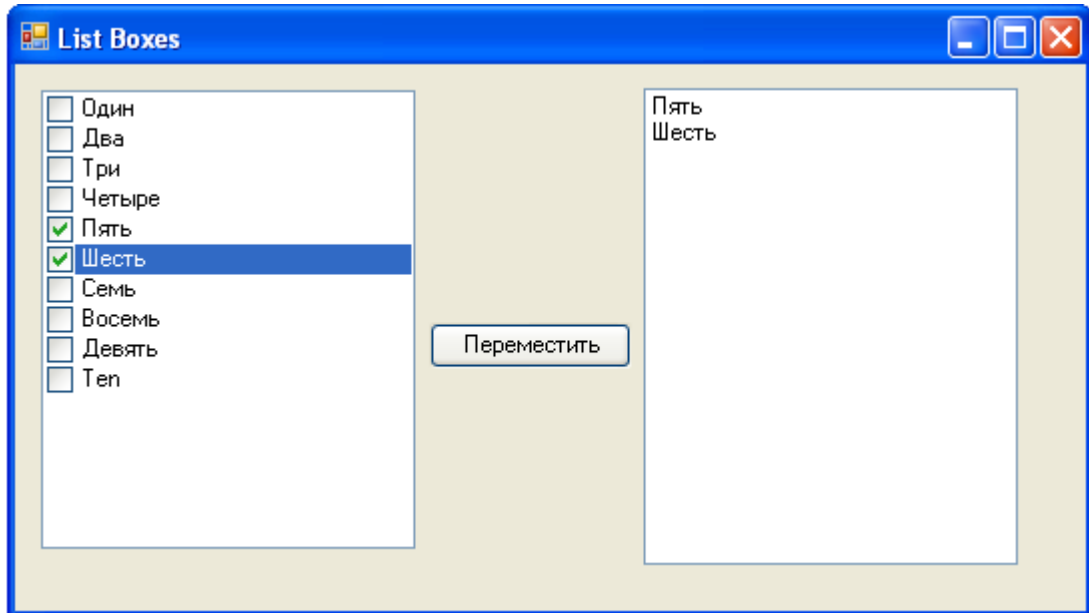


Рис. 3.10. Форма с двумя видами списков.

4. Создайте приложение, на форме которого разместите три полосы прокрутки для управления цветом (красным, зеленым, синим), который используется как фон формы. Начальные положения бегунков полос прокруток должны соответствовать заданному в дизайнера цвету фона формы.

5. Создайте форму (рис. 3.11), которая позволит ввести имя, адрес, род занятий и возраст. На форме должны располагаться кнопки `OK` и `Help`. Данные из формы считываются и заносятся в результирующее текстовое поле, расположенное на этой же форме. При нажатии на кнопку `Help` выводится краткое описание каждого текстового окна в результирующем текстовом поле (рис. 3.12). В программе должна быть реализована проверка вводимого текста по следующим критериям:

- а) поле с именем пользователя не должно быть пустым;
- б) возраст пользователя должен представлять собой число, большее или равное 0;
- в) род занятий пользователя должен описываться как программист или оставаться пустым (можно использовать флажок или текстовое поле);
- г) поле с адресом пользователя не может оставаться пустым;
- д) поле для вывода результирующей информации должно быть только для чтения.

Подсказка: Для проверки значения возраста необходимо обработать сообщение `KeyPress`. Нажатие кнопки `OK` невозможно до введения всей необходимой информации.

Рис. 3.11. Внешний вид формы при нажатии кнопки Help.

Рис. 3.12. Информация из текстовых полей и других элементов управления заносится в результирующее текстовое поле при нажатии кнопки ОК.

5. Создайте простейший редактор текста (рис. 3.13) с использованием элемента RichTextBox. При нажатии на кнопку «Открыть» загружается в расширенное текстовое поле текст из файла формата rtf с некоторым именем (напр., `richTextBoxText.LoadFile("Test.rtf");`), а при нажатии на кнопку «Сохранить» текст из расширенного текстового поля сохраняется в файл с тем же именем. При нажатии на кнопку «По центру» необходимо проверить свойство `SelectionAlignment` текстового поля на предмет того, не является ли выбранный текст уже выровненным по центру. `HorizontalAlignment` (выравнивание по горизонтали) — это перечислимый тип, который может принимать следующие значения: `Left`, `Right`, `Center`, `Justify` (по формату) и `Notset` (не задано). В данном случае необходимо выполнить проверку на предмет того, на равно ли значение `Center`, и если равно, то выполняется выравнивание по левому краю. Если нет — задается выравнивание по центру.

Рис. 3.13. Редактор текста.