Лабораторная работа №4

Тема: Создание приложений по обработке списков данных в средеVisual Studio на языке C#

Цель работы: Исследовать возможностей интегрированной среды разработки Visual C# и получить практические навыки по созданию приложений по обработке списков данных.

Оборудование: IBM – совместимые компьютеры.

Место проведения: Компьютерный класс.

Техника безопасности: См. инструкцию.

Теоретические сведения

1.1. Компонент ListBox

Компонент **ListBox** представляет собой список элементов, из которых пользователь может выбрать как один, так и множество элементов. Компонент создает прямоугольную область, в которой отображается список текстовых строк. Эти текстовые строки можно добавлять в список, выбирать или удалять из него. Свойства компонента приведены в табл. 1.

Таблица 1. Свойства компонента ListBox

Свойство	Описание
Items	Элементы списка — коллекция строк
Count	Количество элементов списка
SelectedIndex	Номер элемента, выбранного в списке. Если ни один из элементов списка не выбран, то значение свойства равно -1
Sorted	Признак необходимости автоматической сортировки (**True**) списка после добавления очередного элемента
SelectionMode	Определяет режим выбора элементов списка: **One** — только один элемент; **MultiSimple** — можно выбрать несколько эле ментов, сделав щелчок на нужных элементах списка; **MultiExtended** — можно выбрать несколько элементов, сде лав щелчок на нужных элементах списка при нажатой клави ше <**Ctrl**>, или выделить диапазон, щелкнув при нажатой кла више <**Shift**> на первом и последнем элементе диапазона
ScrollAlways Visible	Признак необходимости всегда отображать вертикальную полосу прокрутки (значение свойства равно **True**). Если значение свойства равно **False**, то полоса прокрутки отображается, только если все элементы списка нельзя отобразить при заданном размере компонента
MultiColumn	Признак необходимости отображать список в несколько колонок. Количество отображаемых колонок зависит от количества элементов и размера области отображения списка
ColumnWidth	Задает ширину колонки списка при многоколоночном списке.
DataSource	Задает источник данных, с помощью которого можно за полнять список (в данной версии источник данных отключен). Сущест вуют два способа: - использовать метод **Add()** — в этом случае свойство **DataSourse** должно быть отключено; - подключаться к различным источникам данных, доступ к которым формируется через диалоговое окно, открывающееся кнопкой, распо ложенной в поле этого свойства.
Location	Положение компонента на поверхности формы

Size	Размер компонента без (для компонентов типа **DropDown** и **DropDownList**) или с учетом (для компонента типа **Simple**) размера области списка или области ввода
Font	Шрифт, используемый для отображения содержимого поля редактирования и элементов списка

Если множество элементов превосходит размеры окна, то в компоненте автоматически появляется полоса прокрутки.

Если свойство **MultiColumn** установлено в **True**, то компонент выводит элементы в несколько колонок, при этом появляется горизонтальная полоса прокрутки. Если свойство **MultiColumn** установлено в **False**, то вывод элементов идет в одну колонкуи появляется вертикальная полоса прокрутки.

Свойство **SelectedIndex** возвращает целочисленное значение, которое соответствует первому элементу в списке выбранных. Если выборка оказалась пустой, то значение этого свойства устанавливается в -1. Значение индекса в списке изменяется от нуля. При многострочной выборке это свойство возвращает индекс первого элемента из списка выбранных.

Свойство **SelectedItem** возвращает выбранный элемент. Обычно это текстовая строка.

Количество элементов в списке сообщается в свойстве **Count**, значение которого всегда на единицу больше индекса последней строки списка, потому что последний отсчитывается от нуля.

Список, отображаемый в поле компонента, можно сформировать во время создания формы или во время работы программы. Чтобы сформировать спи сок во время создания формы, надо в окне

«**Свойства**» свойство **Items**, щелкнуть на кнопке с тремя точками (она находится в поле значения строки свойства **Items**) и в появившемся окне «**Редактор коллекции строк**» ввести элементы списка.

Чтобы сформировать список во время работы программы, надо применить метод **Add()** к свойству **Items**. Например, следующий фрагмент кода формирует упорядоченный по алфавиту список:

listBox1.Items.Add("рабочий"); listBox1.Items.Add("менеджер");
listBox1.Items.Add("директор"); listBox1.Items.Add("консультант"); listBox1.Sorted = true;

Кроме того, для работы со списком во время выполнения приложения используются следующие методы свойства **Items**:

Insert() — вставить элемент внутрь списка;

Clear() — удалить все элементы из списка (очищает список);

Remove() — удалить заданный элемент из списка. Извлечь строки из компонента можно так:

string Str = listBox1.Items[i].ToString();

где **i** — это номер строки (начинается с нуля).

Обнаружить строку, на которой был щелчок мыши (обработка события **Click**), можно так:

string Str=this.listBox1.Items[this.listBox1.SelectedIndex].ToString();

где **SelectedIndex** — индекс выбранной строки.

Пример 1. Создадим приложение "Колонки", демонстрирующее возможности одноколоночного и многоколоночного режимов компонента **ListBox** (ее форма приведена на рис. 1,2,3, а текст функции обработки событий **Click**, возникающего при щелчке на кнопках «Одноколоночный режим» и

«Многоколоночный режим» — в лис тинге 1).

Листинг 1. Обработка событий Click

private void button1_Click(object sender, EventArgs e)

```
{
listBox1.Items.Clear(); // Очистка компонента
string Str; listBox1.MultiColumn=false;
for (int i=0; i<25; i++)
{
Str=Convert.ToString(i); Str="Строка_"+ Str; listBox1.Items.Add(Str);

}
}
private void button2_Click(object sender, EventArgs e)
{
listBox1.Items.Clear(); // Очистка компонента
for (int i=0; i<25; i++)
{
string Str=Convert.ToString(i); Str = "Строка_" + Str; listBox1.Items.Add(Str);
}
listBox1.MultiColumn=true; // устанавливаем многоколоночный режим
listBox1.ColumnWidth=62; // устанавливаем ширину колонки в пикселях
}
```

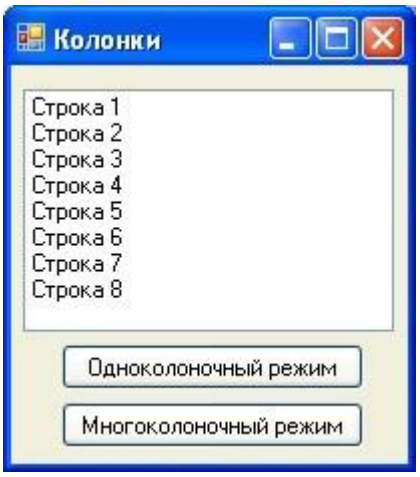

Рис. 1. Вид приложения, строки в котором сформированы с помощью «Редактора»

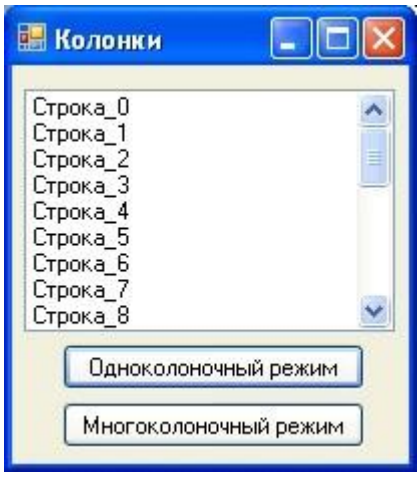

Рис. 2. Вид приложения, строки в котором сформированы приложением «Одноколоночный режим»

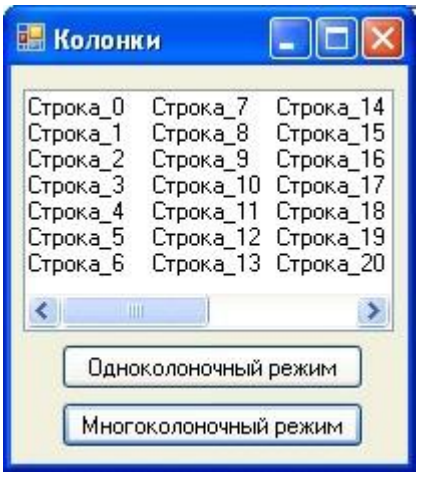

Рис. 3. Вид приложения, строки в котором сформированы приложением «Многоколоночный режим»

Для того чтобы автоматически выводить любой текст в многоколоночном режиме, надо уметь определять строку максимальной длины в пикселях. В общем случае для этого можно воспользоваться нижеприведенным кодом:

```
string Str;
string Str_0 = listBox1.Items[0].ToString(); for (int i = 0; i < listBox1.Items.Count; i++)
{
Str = listBox1.Items[i].ToString(); if (Str.Length > Str_0.Length)
Str_0 = listBox1.Items[i].ToString();
}
int width = (int)listBox1.CreateGraphics().MeasureString(Str_0, listBox1.Font).Width; listBox1.ColumnWidth = width;
```

Метод **CreateGraphics()** создает графический объект для **listBox1** (вывод строк происходит в графике), у которого есть метод **MeasureString(String, Font)**, который измеряет длину строки в пикселях, выводимую данным шрифтом (разные шрифты занимают на экране разное количество пикселей). Параметр **String** самый длинный элемент, параметр **Font** - шрифт элемента.

Метод **MeasureString (String, Font)** возвращает данные по структуре типа **SizeF**, которая состоит из 2-х элементов типа **float**. Это координаты прямоугольника(в пикселах), куда помещается изображение строки на экране, надо знать ширину этого прямоугольника, т. к. она определяет размер в пикселях. Поэтому выбираем из структуры элемент, определяющий ширину **MeasureString(String, Font).width**, и преобразуем значение в **int,** так как этот элемент имеет тип **float**, а ширина в **ListBox** задается в **int**.

1.2. Компонент ComboBox

Компонент **ComboBox** представляет собой комбинацию поля редак тирования и списка **ListBox**: в форме он представляется в виде редактируемого поля с треугольной кнопкой справа. Компонент **ComboBox** используется для вывода данных в виде выпадающего списка и последующей выборки их из этого списка. По умолчанию **ComboBox** появляется в виде окна для ввода/вывода текста (аналог однострокового **TextBox**), при этом выпадающий список скрыт. Он является также аналогом компонента **ListBox**, из которого пользователь может выбрать элементы. Свойства компонента приведены в табл. 2.

Таблица 2. Свойства компонента ComboBox

Свойство	Описание
Items	Содержит набор строк. Это свойство можно как задавать в режиме дизайна, открыв диалоговое окно редактора кнопкой с многоточием в поле этого свойства, чтобы ввести туда необходимые строки, так и программно формировать. Если некоторая строка отмечена в **ComboBox**, то ее индекс помещается в свойство **SelectedIndex**.
Items.Count	Количество элементов списка **ComboBox**. Расчет количества ведется от 1 (если в списке 10 строк, то **Count** будет равен 10).
SelectedIndex	Номер элемента, выбранного в списке. Это целочисленная переменная, изменяющаяся от нуля (т. е. первая строка **ComboBox** будет иметь индекс, равный нулю, вторая — единице и т.т.) Если кому не нравится произносить слово "индекс (указатель)", то мо назвать его просто номером строки с учетом его отсчета от нуля. Можно программно изменять выбранный из **ComboBox** элемент, изменяя значение **SelectedIndex**. Если ни один из элементов списка не выбран, то значение свойства равно -1.

SelectedItem	Свойство, сходное со свойством **SelectedIndex**, только оно возвращает выбранный элемент (обычно это строка).
DropDownStyle	Свойство, задающее стиль вывода данных компонен том. Может принимать значения: **- Simple** (поле ввода без списка) — в этом случае работает поле редактирования, а кнопка рас крытия списка спрятана (можно только вводить строку данных); - **DropDown** - (поле ввода и раскрывающийся список) — в этом случае стрелка раскрытия списка видна, и с ее по мощью можно раскрыть список, выбрать строку, которая попадет в поле редактирования, где ее можно отредактировать, прежде чем ис пользовать далее; - **DropDownList** (рас крывающийся список) — выборку из списка можно делать, но выбранную строку уже редактировать нельзя.
Sorted	Признак необходимости автоматической сортировки (**True**) списка после добавления очередного элемента
Text	Текст, находящийся в поле ввода/редактирования (для компо нентов типа **DropDown** и **Simple**)
DropDownWidth	Ширина выпадающего списка
DropDownHeight	Высота выпадающего списка. Если свойство таково, что окно списка не вмещает весь список, то в окне появится полоса прокрутки. Если же окно по размеру больше списка, то при выводе окно примет размер списка
	.
MaxDropDownItems	Количество отображаемых элементов в раскрытом списке. Если количество элементов списка больше чем **MaxDropDownItems**, то появляется вертикальная полоса прокрутки
FlatStyle	Стиль окна редактирования. Если задать это свойство в виде **PopUp**, то при наведении курсора мыши на окно оно "всплывет", что весь ма удобно при контроле за движением курсора мыши. Если свойству придать значение **System**, то при наведении курсора мыши на окно стрел ка, раскрывающая список, изменит цвет.
FormatString	С помощью этого свойства можно задавать форматы вы вода некоторых типов данных (чтобы задать формат, надо посредством кнопки с многоточием открыть диалоговое окно и выбрать подходящий формат для выводимых строк). При этом надо помнить, что элементы списка должны быть соответствующего типа (датами, данными по валю те и т. п.).
Location	Положение компонента на поверхности формы
Size	Размер компонента без (для компонентов типа **DropDown** и **DropDownList**) или с учетом (для компонента типа **Simple**) размера области списка или области ввода
Font	Шрифт, используемый для отображения содержимого поля редактирования и элементов списка
AutoComplete CustomSourse	Это свойство совместно со свойствами **AutoCompleteMode** и **AutoCompleteSourse** обеспечивает подсказку с выбором значения из списка для вводимых строк.

Список, отображаемый в поле компонента, можно сформировать во время создания формы или во время работы программы аналогично действиям с компонентом **ListBox**.

Компонент **ComboBox** содержит большое количество событий в первую очередь интересует то, которое наступает, если в выпадающем списке щелкнуть на выбранной строке. При этом мы должны будем попасть в обработчик этого события, чтобы в нем извлечь из списка нужную нам строку. Это событие — **DropDownClosed**. Действительно, событие возникает, когда закрывается выпадающий список. А он закрывается именно после щелчка на какой-либо его строке, либо после щелчка на поле редактирования (в этом случае никакой выборки не произведено и поэтому свойство **SelectedIndex** устанавливается в **-1**, следовательно, эту ситуацию в обработчике тоже надо учитывать).

Компонент **ComboBox** имеет большое количество методов, которые можно посмотреть в справочной системе, нажав <**F1**> при подсвеченном в редакторе текста имени этого объекта. Отметим только часто применяемые, такие как:

- **Focus()** — передает фокус ввода компоненту: компонент становится ак тивным и с ним можно работать;
- **Hide()** — делает объект невидимым (прячет его). При загрузки **ComboBox** используются методы:
- **Clear()** — очищает поле **ComboBox**.
- **Add()** — добавляет элемент в конец поля **ComboBox**.
- **Insert(номер_строки, "текст")** — вставляет строку в поле **ComboBox** перед указанным в аргументе индексом. Например, следующие операторы, помещенные в обработчик кнопки,

comboBox1.Items.Clear();

comboBox1.Items.Add("директор"); *//индекс этой строки равен 0* ***comboBox1.Items.Add("менеджер");*** *//индекс этой строки равен 1* ***comboBox1.Items.Insert (1, "консультант");***

дают такой результат:

директор

консультант менеджер

- **IndexOf()** — выдает индекс строки в поле **ComboBox**. Строка задается в аргументе метода. Если строка не найдена, то выдается **-1**.Индекс в **ComboBox** изменяется от нуля. Это можно проверить методом **IndexOf()**, выполнив после указанных выше операторов оператор:

int i=comboBox1.Items.IndexOf("директор");

Значение **i** будет равно **0**.

- **Remove() -** удаляет строку, указанную в аргументе, из поля **ComboBox**. Например, выполнив оператор на множестве трех предыдущих строк:

comboBox1.Items.Remove ("консультант");

получим результат:

директор менеджер

- **RemoveAt(номер строки)** — удаляет строку из поля **ComboBox**, индекс которой указан в аргументе. Например, выполнив оператор на множестве трех предыдущих строк:

comboBox1.Items.RemoveAt (2);

получим такой результат: **директор консультант**

Для работы с **ComboBox** широко применяется свойство **Count** из **Items.**В нем всегда находится количество элементов **ComboBox**. Например, если выполнить оператор:

int i=comboBox1.Items.Count;

то в переменной **i** получим количество строк в **ComboBox**.

Пример 2. Создадим приложение "Жалюзи", демонстрирующее возможности компонента **ComboBox** для ввода данных (ее форма приведена на рис. 4, результат выполнения на рис. 5, а текст функции обработки событий **Click**, возникающего при щелчке на кнопке — в лис тинге 2).

Настройку компонента **ComboBox** выполняет конструктор формы. Свой ству **SelectedIndex** конструктор присваивает значение "**-1**". Поэтому при появлении формы на экране название материала не отображается. Кроме того, кнопка **ОК** становится доступной только после выбора материала и ввода размеров жалюзи. Доступностью кнопки управляет функция обработки события **TextChanged** полей редактирования (одна функция обрабатывает со бытие **TextChanged** обоих компонентов).

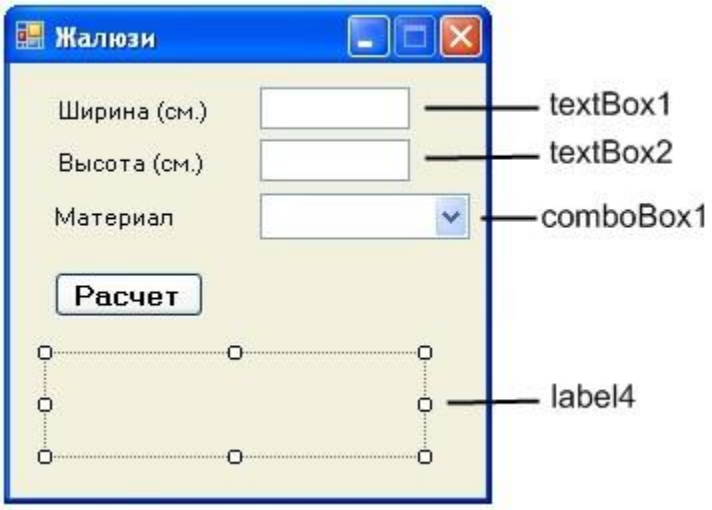

Рис. 4. Форма приложения «Жалюзи»

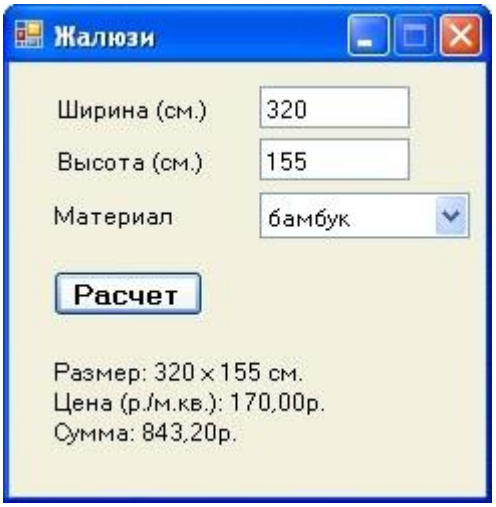

Рис. 5. Результат выполнения приложения «Жалюзи»

Листинг 2. Обработка приложения «Жалюзи»

```
public Form1()
{
InitializeComponent();
// настройка компонентов
comboBox1.DropDownStyle = ComboBoxStyle.DropDownList;
comboBox1.Items.Add("пластик"); comboBox1.Items.Add("алюминий");
comboBox1.Items.Add("бамбук"); comboBox1.Items.Add("соломка");
comboBox1.Items.Add("текстиль"); comboBox1.SelectedIndex = -1;
}

private void button1_Click(object sender, EventArgs e)
{
double w, h, sum;
double cena = 0; // цена за 1 кв.м.
w = Convert.ToDouble(textBox1.Text); h = Convert.ToDouble(textBox2.Text);
switch (comboBox1.SelectedIndex)
{
// элементы списка нумеруются с нуля case 0: cena = 100; break; // пластик case 1:
cena = 250; break; // алюминий case 2: cena = 170; break; // бамбук case 3: cena = 170;
break; // соломка case 4: cena = 120; break; // текстиль
}
sum = (w * h) / 10000 * cena;
label4.Text = "Размер: " + w + " x " + h + " см.\n" + "Цена (р./м.кв.): " +
cena.ToString("c") + "\nСумма: " + sum.ToString("c");
}
```

```
// Метод обрабатывает событие KeyPress компонентов textBox1 и textBox2
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
if ((e.KeyChar >= '0') && (e.KeyChar <= '9')) return;
if (Char.IsControl(e.KeyChar))
{
if (e.KeyChar == (char)Keys.Enter)
{
if (sender.Equals(textBox1)) // клавиша <Enter> нажата в поле Ширина
textBox2.Focus();                    // переместить курсор в поле Высота
else
// клавиша <Enter> нажата в поле Высота button1.Focus(); // переместить
фокус на comboBox1
}
return;
}
e.Handled = true; // остальные символы запрещены
}

// в списке Материал пользователь выбрал другой элемент
private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
{
if ((textBox1.Text.Length != 0) && (textBox2.Text.Length != 0)) button1.Enabled =
true;
label4.Text = "";
}
// пользователь изменил размер (содержимое textBox1 или textBox2)
private void textBox1_TextChanged(object sender, EventArgs e)
{
label4.Text = "";
if ((textBox1.Text.Length == 0) || (textBox2.Text.Length == 0) ||
comboBox1.SelectedIndex == -1 )
button1.Enabled = false; else
button1.Enabled = true;
}
```

Пример сохранения строк из ListBox в файл:

```
private void button1_Click(object sender, EventArgs e)
        {
           using (System.IO.StreamWriter sw = new
System.IO.StreamWriter("C:\\название_файла.txt"))
           {
              for (int i = 0; i < listBox1.Items.Count; i++)
                 sw.WriteLine(listBox1.Items[i].ToString());
           }
        }
```

Пример загрузки строк из файла в ListBox:

```
private void button2_Click(object sender, EventArgs e)
        {
```

```
            listBox1.Items.Clear();
            using (System.IO.StreamReader sr = new
System.IO.StreamReader("C:\\название_файла.txt"))
            {
               while(!sr.EndOfStream)
               {
                  listBox1.Items.Add(sr.ReadLine());
               }
            }
}
```

Задание 1.

Разработать программу «Обработка строк» (Рисунок 6), которая должна выполнять следующие функции:

- по щелчку на кнопке «Добавить строку» записывать в «Список студентов» строку из «Поля ввода», содержащую фамилию и инициалы студента. 20 фамилий можно записать с помощью «Редактора коллекции строк»;
- по щелчку на кнопке «Вставить в список» вставлять перед выделенной строкой в «Списке студентов» строку из «Поле ввода»;
- по щелчку на кнопке «Изменить строку» изменять содержимое выделенной строки в «Списке студентов» на содержимое из «Поле ввода»;
- по щелчку на кнопке «Удалить из списка» удалять строку из «Списка студентов»;
- по щелчку на кнопке «Очистить список» - удаление всех данных из списка;
- кнопки «Открыть» и «Сохранить» предназначены для открытия и сохранения данных в файле.

При разработке интерфейса приложения использовать компоненты **Label, Button, TextBox** и **ListBox** (один из вариантов интерфейса представлен на рис. 6).

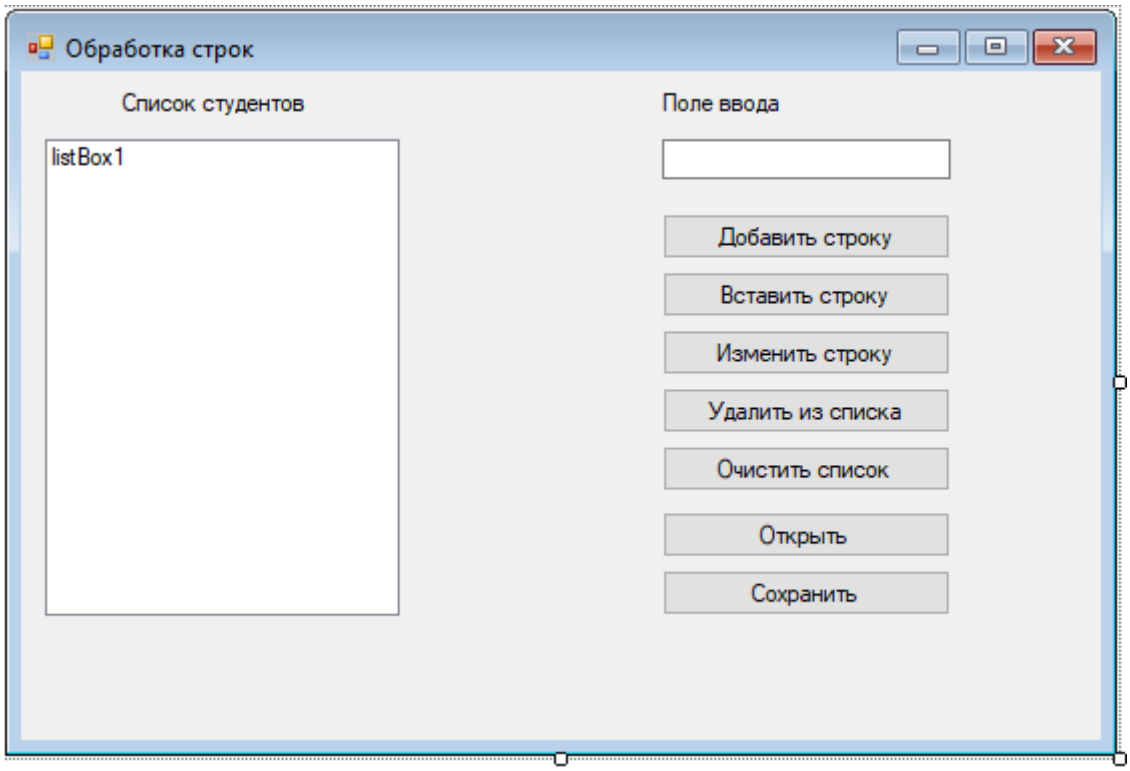

Рис. 6. Внешний вид приложения «Обработка строк»

Задание 2.

1. Установить на форму компонент главного меню, в котором продублировать выполнение функций зафиксированных за кнопками.
2. Добавить в меню открытие формы О программе.