Лабораторная работа «Создание главного и контекстного меню»

Цель: Овладение навыками создания меню различного типа и обработчиков сообщений пунктов меню.

1. Теоретическая часть. Основные сведения о меню.

Обычно программы в среде Windows имеют главное меню, которое образует более или менее сложную структуру выполняемых приложением команд. Создание главного меню приложения является фактически альтернативой выполнения команд элементами управления на форме в соответствии с программным кодом процедур, написанных для этих элементов.

Однако, в отличие от элементов управления, которые осуществляют выполнение различных процедур при совершении различных событий над элементом управления, главное меню приложения позволяет создавать иерархию вложенных друг в друга меню команд любой степени сложности. Особенно эффективно создание главного меню приложения в том случае, когда в приложении необходимо выполнять множество различных команд. При этом команды, выполняющиеся из главного меню, будут собраны в одной строке главного меню, которое можно раскрыть в любой нужный момент для выполнения требуемой команды, не занимая много места в окне приложения на экране дисплея, где должна происходить основная обработка информации.

Главное и контекстное меню — это абсолютно различные вещи с точки зрения функционального назначения. В главное меню выносят все функции, которые выполняет программа. В любой момент пользователь может воспользоваться нужным пунктом меню для совершения какого-либо действия. С контекстным меню все по-другому. Оно должно включать лишь те пункты, которые соответствуют позиции вызова контекстного меню. Контекстное меню появляется при нажатии правой кнопки мыши.

2. Практическая часть.

2.1. Создание главного меню.

Для создания главного меню приложения Visual Studio .NET имеет в панели ToolBox компонент MenuStrip. Создайте новое Windows Application C# приложение с именем MenuApp. Добавьте на форму компонент MenuStrip. В панели компонентов ниже основной формы приложения появится объект menuStrip1. В верхней части формы появится проект меню с единственным полем «Type Here».

Поле является редактируемым, если вы измените надпись в поле, то справа и снизу от него появятся дополнительные поля (рис. 4.1). Добавьте в меню пункты так, как показано на рис. 4.1.

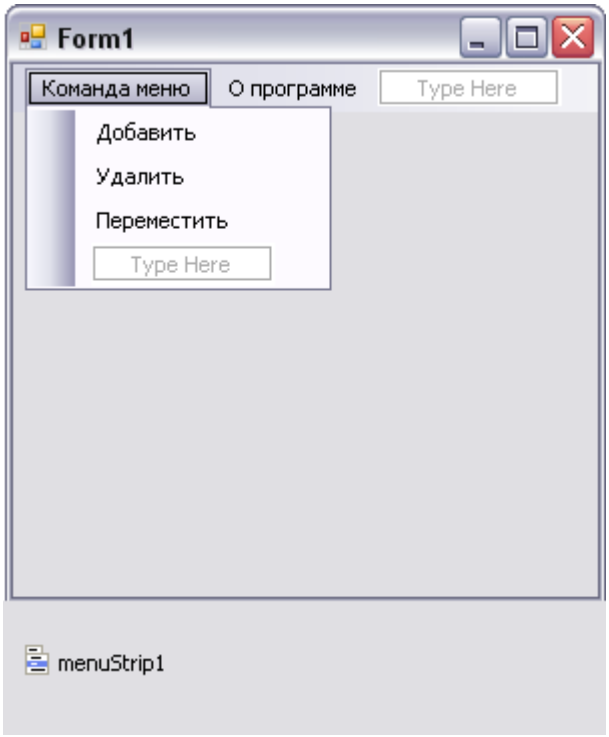

Рис. 4.1. Добавление пунктов меню.

Пункт меню «Команда меню» содержит три подпункта:

• Добавить
• Удалить
• Переместить.

Кроме пункта «Команда меню» основное меню содержит пункт «О программе». Такой пункт обычно присутствует во всех коммерческих приложениях и предоставляет пользователю информацию о версии программы, разработчиках и т. д. Пункт меню «О программе» не имеет подпунктов. Для лучшей читаемости программы измените свойство Name каждого пункта меню.

Команда меню – menuItemCommand;
Добавить – menuItemAdd;
Удалить – menuItemDel;
Переместить – menuItemMove;
О программе – menuItemAbout.

Компонент MenuStrip представлен в коде программы классом System.Windows.Forms.MenuStrip.

```
private System.Windows.Forms.MenuStrip menuStrip1;
```

Каждому пункту меню в коде программы соответствует объект класса ToolStripMenuItem.

```
    private          System.Windows.Forms.ToolStripMenuItem
menuItemCommand;
    private          System.Windows.Forms.ToolStripMenuItem
menuItemAdd;
    private          System.Windows.Forms.ToolStripMenuItem
menuItemDel;
    private          System.Windows.Forms.ToolStripMenuItem
menuItemMove;
    private          System.Windows.Forms.ToolStripMenuItem
```

```
menuItemAbout;
```

Вот как происходит добавление элементов главного меню:

```
    this.menuStrip1.Items.AddRange(new
System.Windows.Forms.ToolStripItem[]
    {     this.menuItemCommand,
      this.menuItemAbout});
```

Эти строки кода добавляют в объект menuStrip1 два пункта меню: «Команда меню» и «О программе». Свойство Items объекта MenuStrip имеет функцию AddRange, которая добавляет массив элементов ToolStripItem[] в меню.

```
    this.MenuItemCommand.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
                this.menuItemAdd,
                this.menuItemDel,
                this.menuItemMove});
```

Затем, в пункт меню menuItemCommand («Команда меню») добавляется три пункта menuItemAdd, menuItemDel, menuItemMove. Теперь становится все очень понятно: для того чтобы внести изменения в какой-либо пункт меню, вам необходимо будет изменить соответствующий объект класса ToolStripItem.

2.2. Создание вложенного меню.

Разработчики C# максимально упростили работу с меню. Поэтому создание вложенного меню не вызовет у вас затруднений. Вы уже создавали вложенное меню, когда добавляли в пункт меню «Команда меню» несколько пунктов. Но там вы помещали подпункты в главное меню приложения. Подпункты меню тоже могут содержать вложенные элементы. Для того чтобы «вложить» элемент в пункт меню, необходимо добавить название в поле «Type Here», находящееся справа от пункта меню (рис. 4.2).

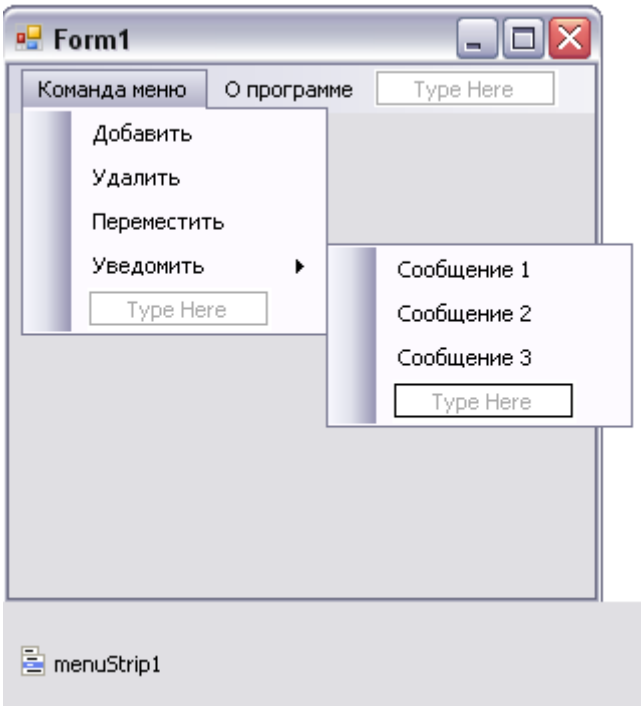

Рис. 4. 2. Создание вложенного меню.

При этом с правой стороны пункта меню появится указательная стрелка, свидетельствующая о наличии вложенных пунктов меню. Добавьте в наше приложение пункт меню «Уведомить» и вложите в него пункты «Сообщение 1»,

«Сообщение 2», «Сообщение 3» так, как показано на рис. 4.2.

2.3. Обработка сообщений меню.

Основным сообщением пункта меню является Click. Это сообщение приходит, когда пользователь выбирает пункт меню. Давайте добавим обработчики к пунктам меню. Для этого необходимо щелкнуть два раза по полю с именем сообщения в окне свойств пункта меню (рис. 4.3).

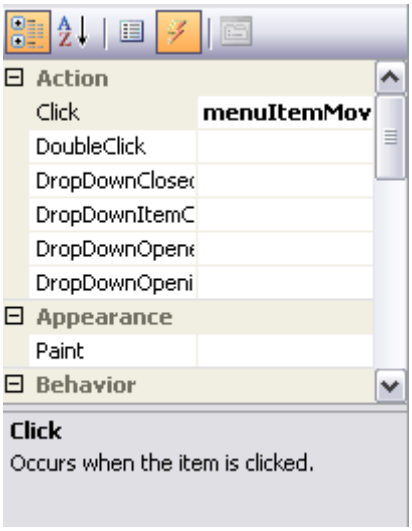

Рис. 4.3. Окно свойств элемента MenuItem. Закладка событий.

Добавьте обработчики для пунктов меню «Добавить», «Удалить» и «Переместить». При этом в код программы добавятся три новых метода. Измените код этих методов так, как показано ниже:

```
    private void menuItemAdd_Click(object sender,
System.EventArgs e)
            {// код для добавления
                MessageBox.Show("Добавление");
            }
            private void menuItemDel_Click(object sender,
System.EventArgs e)
            {// код для удаления
                MessageBox.Show("Удаление");
            }

            private void menuItemMove_Click(object sender,
System.EventArgs e)
            {// код для перемещения
                MessageBox.Show("Перемещение");
            }
```

Теперь при выборе одного из указанных выше пунктов меню на экране появится соответствующее сообщение. Кроме того, C# позволяет нескольким пунктам меню использовать один и тот же обработчик сообщения. Для того чтобы поэкспериментировать с такой возможностью, выделите сразу три пункта меню: «Сообщение 1», «Сообщение 2», «Сообщение 3», удерживая нажатой клавишу Ctrl. Теперь в окне свойств щелкните два раза по событию Click. Таким образом, вы присвоите всем трем пунктам меню один и тот же обработчик. Посмотрите, как это выглядит в коде программы:

```
        this.сообщение1ToolStripMenuItem.Click += new
System.EventHandler(this.сообщение1ToolStripMenuItem_Click);
        this.сообщение2ToolStripMenuItem.Click += new
System.EventHandler(this.сообщение1ToolStripMenuItem_Click);
    this.сообщение3ToolStripMenuItem.Click += new
System.EventHandler(this.сообщение1ToolStripMenuItem_Click);
```

Для всех трех событий различных пунктов меню устанавливается один и тот же обработчик — сообщение1ToolStripMenuItem_Click. Напишите следующий код для функции сообщение1ToolStripMenuItem_Click:

```
    private void сообщение1ToolStripMenuItem_Click(object
sender, EventArgs e)
            {              ToolStripMenuItem item =
(ToolStripMenuItem)sender;
                string message = item.Text;
                MessageBox.Show(message);   }
```

Один из параметров любого обработчика события object sender — это объект, который послал сообщение. Поскольку все объекты в C# являются наследниками класса System.Object, то использование класса object в качестве параметра «универсализирует» использование обработчиков событий. В данном случае обработчик сообщение1ToolStripMenuItem_Click() получает события только от объектов класса ToolStripMenuItem. Поэтому можно смело приводить объект sender к классу ToolStripMenuItem.

```
                ToolStripMenuItem item =
(ToolStripMenuItem)sender;
```

Для того чтобы различить, от какого именно пункта меню пришло событие, считывается значение свойства Text пункта меню.

```
    string message = item.Text;
```

Это свойство различно у всех пунктов меню. Поэтому можно вычислить, какой именно объект прислал сообщение. В данном примере просто выводится на экран сообщение с текстом пункта меню.

```
        MessageBox.Show(message);
```

2.4. Создание контекстного меню.

Поместите на форму приложения MenuApp компонент TrackBar. Расположите его по своему усмотрению. В главное меню приложения между пунктами «Команда меню» и «О программе» добавьте пункт «Стиль бегунка». Вложите в этот пункт четыре подпункта: «Пусто», «Сверху-слева», «Снизу-справа», «С обеих сторон» (рис. 4.4).

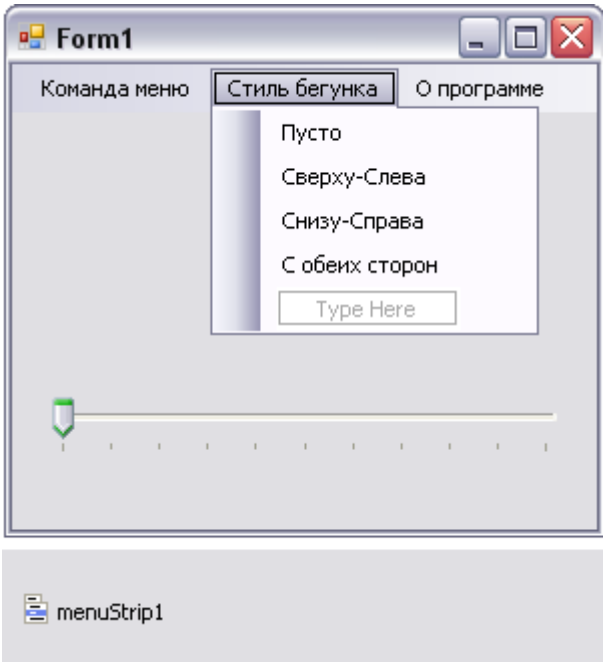

Рис. 4.4. Главное меню программы управления бегунком.

Переименуйте пункты меню, изменив свойство Name каждого элемента:

- `Стиль бегунка – menuItemTrackBar;`
- `Пусто – menuItemNone;`
- `Сверху -слева – menuItemTopLeft;`
- `Снизу -справа – menuItemBottomRight;`
- `С обеих сторон – menuItemBoth.`

Для всех вышеописанных элементов меню присвойте один и тот же обработчик сообщения. Для этого выделите по очереди все четыре пункта меню, удерживая нажатой кнопку Ctrl. В окне свойств, на закладке Properties, щелкните два раза указателем мыши по событию Click. При этом ко всем пунктам меню добавится обработчик события menuItemNone_Click:

```
    this.menuItemNone.Click                    +=                    new
System.EventHandler(this.menuItemNone_Click);
this.menuItemTopLeft.Click                    +=                    new
System.EventHandler(this.menuItemNone_Click);
this.menuItemBottomRight.Click+=                                    new
System.EventHandler(this.menuItemNone_Click);
    this.menuItemBoth.Click+=                                       new
System.EventHandler(this.menuItemNone_Click);
```

В конец кода программы добавится тело самой функции. Теперь давайте создадим два контекстных меню. Одно из них будет частично дублировать пункт основного меню «Команда меню», другое — пункт «Стиль бегунка». Для этого поместите на форму два компонента ContextMenuStrip: contextMenuStrip1 и contextMenuStrip2. Выберите в панели компонент программы объект contextMenu1. При этом в верхней части формы появится редактируемое поле. Выделите его указателем мыши. Ниже выделенного поля появится поле ввода с надписью «Type Here». Добавьте в контекстное меню пункты «Добавить», «Удалить» и «Переместить». Это контекстное меню будет соответствовать пункту основного меню «Команда меню».

Теперь необходимо присвоить обработчики всем добавленным пунктам меню. Для пункта контекстного меню «Добавить» выберите из списка обработчиков события Click функцию menuItemAdd_Click (рис. 4.5).

Таким образом, событию Click пункта контекстного меню «Добавить» присвоится обработчик события Click пункта основного меню «Добавить».

```
this.добавитьToolStripMenuItem.Click+=
new System.EventHandler(this.menuItemAdd_Click);
```

Тело функции-обработчика уже существует, поэтому при выборе любого из этих двух пунктов меню будет выполняться одно и то же действие. Установите для пунктов контекстного меню contextMenuStrip1 «Удалить» и «Переместить» обработчики menuItemDel_Click и menuItemMove_Click соответственно. После того необходимо установить contextMenuStrip1 контекстным меню для формы Form1. Для этого у объекта Form1 существует свойство ContextMenuStrip. Среда сама добавит в список ContextMenuStrip все необходимые элементы, которые могут быть присвоены этому свойству. В нашем случае список будет состоять из двух элементов (рис. 4.6).

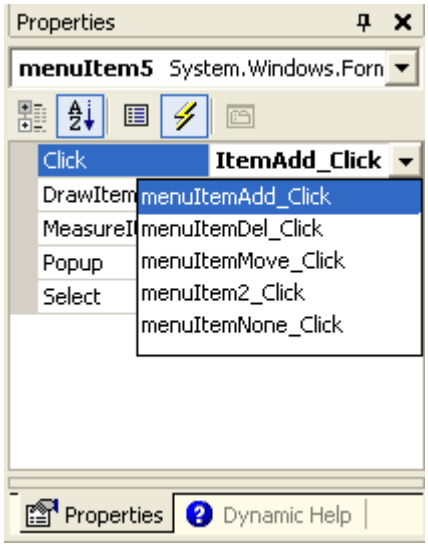

Рис. 4.5. Доступные обработчики событий.

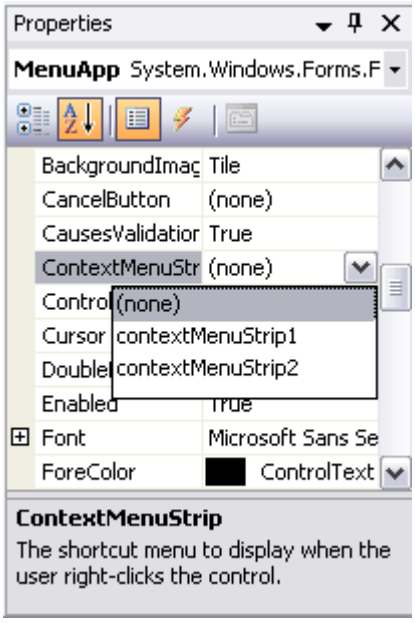

Рис. 4.6. Список доступных контекстных меню.

Установите значение свойства ContextMenuStrip в contextMenuStrip1. Откомпилируйте и запустите программу. Щелкните правой кнопкой мыши по любому свободному месту на форме. На экране появится контекстное меню, которое вы создали (рис. 4.7).

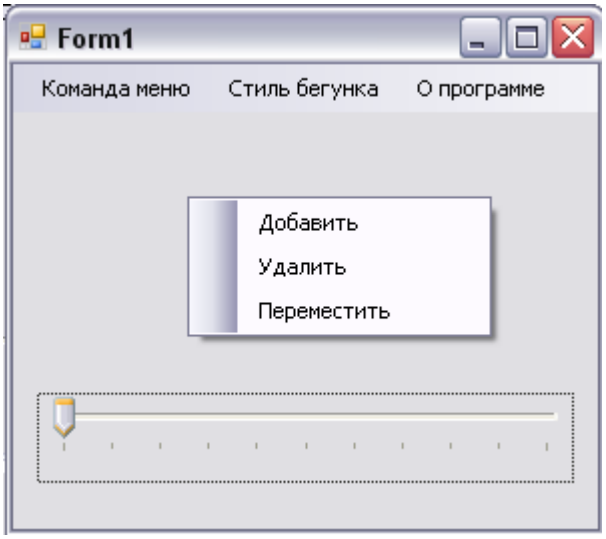

Рис. 4.7. Контекстное меню основной формы программы.

Если вы выберите любой из пунктов контекстного меню, выполнится то же действие, которое соответствует аналогичным пунктам основного меню. Выберите пункт «Удалить» контекстного меню. На экране появится сообщение, изображенное на рис. 4.8. Такое же сообщение появится, если вы выберете пункт «Удалить» основного меню.

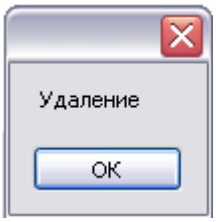

Рис. 4.8. Результат команды «Удалить».

Контекстное меню — это удобная возможность выполнять различные действия над объектами, не загромождая основное окно программы. Допустим, нам необходимо в программе изменять свойства элемента TrackBar. Предоставим пользователю возможность делать это как из основного меню программы, так и из контекстного меню. В основном меню программы у нас уже имеется пункт «Стиль бегунка», который содержит подпункты с названиями стилей. Давайте добавим в контекстное меню contextMenuStrip2 те же подпункты. Для этого выделите объект contextMenuStrip2 и добавьте в него поля: «Пусто», «Сверху-слева», «Снизу-справа», «С обеих сторон».

Теперь добавьте обработчик события Click для пунктов контекстного меню. Для этого выделите все четыре пункта меню, удерживая клавишу Ctrl, и выберите из предложенного списка для события Click обработчик menuItemNone_Click. Теперь все пункты contextMenuStrip2 и соответствующие пункты основного меню имеют один и тот же обработчик — menuItemNone_Click.

Добавьте к этому обработчику код, который представлен ниже:

```
    private void menuItemNone_Click(object sender,
System.EventArgs e)
            {
                ToolStripMenuItem item =
(ToolStripMenuItem)sender;
                string text = item.Text;
                switch(text)
```

```
                {
                    case "Пусто":
                        trackBar1.TickStyle                    =
TickStyle.None;
                        break;
                    case "Сверху-Слева":
                        trackBar1.TickStyle                    =
TickStyle.TopLeft;
                        break;
                    case "Снизу-Справа":
                    trackBar1.TickStyle                        =
TickStyle.BottomRight;
                        break;
                    case "С обеих сторон":
                        trackBar1.TickStyle                    =
TickStyle.Both;
                        break;
                }
            }
```

Для завершения функциональности программы установите свойство ContextMenuStrip компонента trackBarl как contextMenuStrip2. Все, программа закончена. Откомпилируйте и запустите программу. Щелкните правой кнопкой мыши по изображению бегунка на форме.

Вместо контекстного меню, которое появляется при нажатии правой кнопкой мыши на форме, вы увидите контекстное меню, соответствующее управлению стилями бегунка (рис. 4.9).

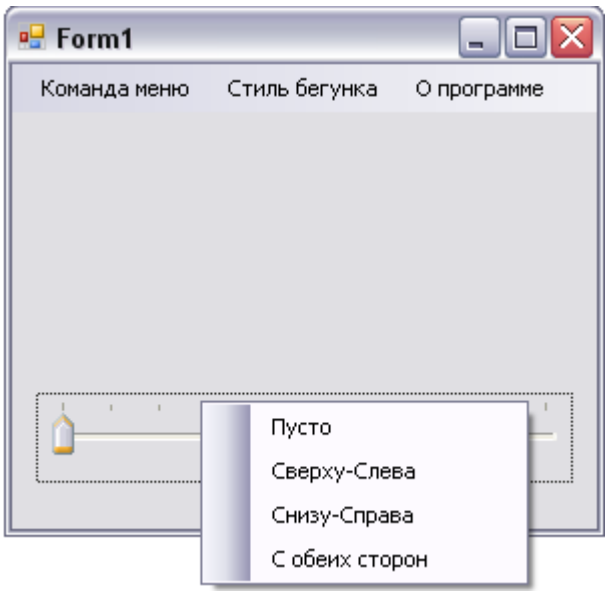

Рис. 4.9. Контекстное меню бегунка.

Попробуйте изменить стиль бегунка путем выбора различных пунктов контекстного меню. Местоположение черточек бегунка будет изменяться в зависимости от выбранного пункта меню. Те же самые операции можно выполнить из основного меню приложения. Давайте детальнее рассмотрим код функции-обработчика.

```
        ToolStripMenuItem item = (ToolStripMenuItem)sender;
```

Присваиваем переменной item объект sender, от которого пришло сообщение. Поскольку заранее известно, что сообщения всегда посылаются от пунктов меню, выполняем приведение типа к ToolStripMenuItem. Объекты класса ToolStripMenuItem имеют свойство Text, которое в нашем случае характеризует пункт меню. Инструкция switch в языке C# поддерживает использование строк в качестве элементов для сравнения. Поэтому можно получить значения свойства Text и по нему идентифицировать пункт меню.

```
string text = item.Text;
switch(text)
//…
```

Далее выполняются несколько инструкций case, и по тексту пункта меню выставляется стиль элемента управления trackBar.

2.5. Пометка пунктов меню.

Очень удобная возможность пунктов меню — их пометка. Обычно это флажок слева от надписи пункта меню. Для пометки пункта меню используется свойство Checked класса ToolStripMenuItem. Это свойство типа bool. Если значение Checked установлено в true, то флажок присутствует, если в false — то отсутствует. Давайте допишем программу так, чтобы текущий стиль элемента управления trackBarl в меню всегда был отмечен флажком. Для этого необходимо изменить код программы. Поместите в конец функции menuItemNone_Click следующие строки кода:

```
// проходим по всем подпунктам изменяющим стиль бегунка
                // расположенным в основном меню программы
    foreach (ToolStripMenuItem item1 in
menuItemTrackBar.DropDownItems)
                {
                  // если текст меню совпадает с переданным
параметром,
                    // то помечаем пункт меню
                    if (item1.Text == text)
                        item1.Checked = true;
                // если текст меню не совпадает с переданным
параметром,
                    // то снимаем пометку с пункта меню
                    else
                        item1.Checked = false;
                }
                // проходим по  всем подпунктам, изменяющим
стиль бегунка и
                //   расположенных   в   контекстном   меню
программы
            foreach (ToolStripMenuItem item1 in
contextMenuStrip2.Items)
                {
                 // если текст меню совпадает с переданным
параметром,
```

```
                // то помечаем пункт меню
                if (item1.Text == text)
                    item1.Checked = true;
            // если текст меню не совпадает с переданным
параметром,
                // то снимаем пометку с пункта меню
                else
                    item1.Checked = false;
            }
```

Для того чтобы программа с самого начала правильно функционировала, установите для пунктов меню «Снизу-Справа» свойство Checked в True по умолчанию, потому как именно этот стиль установлен по умолчанию для объекта trackBar1. Вы можете сделать это, выбрав нужный пункт меню и установив для него свойство Checked как True в окне свойств.

Запустите программу. Выберите в контекстном меню стиль «С обеих сторон». Откройте еще раз контекстное меню — пункт меню «С обеих сторон» будет помечен. Можете изменить стиль на любой другой, пометка всегда будет соответствовать выбранному стилю. То же самое произойдет и с пунктами основного меню: их пометка будет строго соответствовать пунктам контекстного меню.

3. Задания.

1. Создайте пример приложения, описанного в практической части лабораторной работы.

2. Добавьте в приложение пункт меню «Ориентация» с двумя подпунктами «Горизонтальная» и «Вертикальная». С помощью данных пунктов изменяется расположение элемента trackbar1. Аналогичные команды добавьте и в контекстное меню для элемента trackbar1. Пометка пунктов меню должна соответствовать выбранному стилю элемента управления.

Лабораторная работа «Создание многооконных приложений»

Цель: Овладение навыками создания и использования родительских и дочерних форм.

1. Теоретическая часть. Основные сведения о многооконных приложениях.

Традиционно существует три разновидности приложений, а именно:

• **Приложения, основанные на диалоговом окне,** — это приложения в виде единого диалогового окна, с помощью которого может быть осуществлен доступ ко всем функциональным возможностям.

• **Однодокументные интерфейсы (Single Document Interface, SDI)** — это приложения, содержащие меню, одну или несколько линеек инструментов и одно окно, в котором пользователь может выполнять определенные действия.

• Многодокументные интерфейсы (Multi-Document Interface, MDI) похожи на SDI-приложения, однако обладают способностью одновременно поддерживать несколько открытых окон.

Приложения, основанные на диалоговом окне, обычно представляют собой небольшие одноцелевые приложения, которые ориентированы либо для решения конкретной задачи, требующей ввода небольшого количества данных, либо для работы с какими-то необычными типами данных. В качестве примера такого приложения можно привести Calculator (калькулятор), поставляемый вместе с MS Windows.

Однодокументные интерфейсы, как правило, предназначаются для решения какой-то одной конкретной задачи, при этом они позволяют пользователю загружать в приложение единственный документ, с которым он и будет вести работу. Эта задача предполагает выполнение пользователем большого количества действий, и зачастую пользователю могут потребоваться возможности, позволяющие сохранять или загружать плоды своего труда. Хорошим примером SDI-приложений могут служить MS Paint и WordPad, также поставляемые совместно с MS Windows. Однако такие приложения допускают открытие только одного документа в каждый конкретный момент времени, поэтому если пользователю требуется открыть второй документ, то ему будет необходимо открывать новый экземпляр SDI-приложения, у которого будет отсутствовать связи с первым документом и, следовательно, конфигурация, созданная для первого экземпляра, не окажет никакого влияния на конфигурацию второго. Например, вы в MS Paint выбрали красный цвет в качестве цвета рисования, затем открываете второй экземпляр MS Paint, а здесь в качестве цвета, используемого для рисования, выбирается цвет по умолчанию. Он будет черный.

Многодокументные интерфейсы почти полностью аналогичны SDI-приложениям за исключением того, что они обладают возможностью поддерживать более одного открытого документа в различных окнах, которые могут быть открыты одновременно. Одним из простых признаков MDI-приложения является наличие пункта **Окно (Window)** перед пунктом **Справка (Help).** Пункт **Окно** содержит подпункты для переключения между окнами и документами. Примерами MDI-приложений служат Adobe Acrobat Reader и MS Word.

2. Теоретическая часть.

2.1. Создание многооконного приложения.

Для создания MDI-приложения, во-первых, необходимо, чтобы решаемая пользователем задача требовала одновременно несколько открытых документов. Примером задач такого рода является текстовый редактор или программа просмотра документов. Во-вторых, необходимо предусмотреть пункт меню **Окно**, который позволял бы пользователю изменять положение открытых окон друг относительно друга (налагая их друг на друга, например каскадно) и предоставлял бы ему список всех открытых окон. Еще одной особенностью MDI-приложений является то, что если имеется некоторое открытое окно и в этом окне существует некоторое меню, то оно должно быть интегрировано в основное меню приложения.

Любое MDI-приложение состоит по крайней мере из двух разных окон. Первое окно называется **MDI-контейнером (или родительской формой),** а окно, которое может быть открыто в контейнере, называется **дочерним MDI-окном (или просто дочерней формой).** Родительская форма может содержать несколько дочерних окон. Только одно из дочерних окон может быть активно в один момент времени.

При создании MDI-приложения следует начинать с того же, с чего начинается создание любого другого приложения,— с создания Windows Application в Visual Studio. Чтобы превратить основное окно приложения из формы в MDI-контейнер, достаточно просто присвоить свойству формы IsMdiContainer значение true. При этом произойдет изменение цвета фона, указывающее на то, там не следует размещать видимые управляющие элементы, хотя такая возможность по-прежнему существует и может при определенных обстоятельствах оказаться полезной. Для создания дочернего окна следует добавить в проект новую форму, выбрав Windows Form из диалогового окна, которое открывается при выборе пункта меню Project | Add New Item. Эта форма становится дочерним окном, когда его свойству MdiParent присваивается ссылка на основное окно. Этому свойству нельзя присваивать значение с помощью панели Properties, это необходимо выполнять программным путем.

До того, как появится возможность выводить MDI-приложение на экран в его основном виде, нужно выполнить еще две вещи. Необходимо передать MDI-контейнеру информацию о том, какие окна должны выводиться, а затем вывести их, для чего следует просто создать новый экземпляр формы, которую вы собираетесь выводить, а затем вызвать для нее метод show(). Конструктор формы, предназначенной для вывода в качестве дочернего окна, должен привязаться к родительскому контейнеру. Это достигается за счет присваивания его свойства MdiParent экземпляру MDI-контейнера.

Прежде чем переходить к выполнению более сложных задач, разберем небольшой пример, в котором выполняются все эти шаги.

1. Создайте новое Windows Application и назовите его MDIBasic.
2. Выберите форму и задайте ей следующие свойства:

Name ***frmContainer***
IsMdiContainer ***True***

Text ***Родительская форма***
WindowState ***Maximized***

3. Добавьте новую форму к решению, выбрав пункт Windows Form из меню Project | Add New Item (тип добавляемого элемента Windows Form). Назовите эту форму frmChild. Весь код, необходимый для того, чтобы вывести дочернюю форму, располагается в конструкторах форм. Сначала мы займемся конструктором дочернего окна:

```
public frmChild (MDIBasic.frmContainer parent)
                {
                    InitializeComponent();
                    //  Присваивание  контейнеру  родителя
данной формы
                    this.MdiParent = parent;
                }
```

Чтобы привязать дочернюю форму к MDI-контейнеру, ее необходимо зарегистрировать в контейнере. Это достигается присваиванием свойству MdiParent данной формы соответствующего значения, как показано в приведенном выше коде. Заметьте, что используемый конструктор включает в себя параметр parent. Поскольку в C# не предусмотрены конструкторы по умолчанию для класса, в котором описывается его собственный конструктор, приведенный выше код позволяет предотвратить создание экземпляра данной формы, не привязанного к MDI-контейнеру.

Выполним вывод формы на экран в конструкторе MDI-контейнера:

```
  public frmContainer()
            {
                InitializeComponent();
                // Создание нового экземпляра дочерней формы
             MDIBasic.frmChild       child       =       new
MDIBasic.frmChild(this);
                // Вывод созданной формы
                child.Show();
            }
```

Мы создаем новый экземпляр дочернего класса и передаем его конструктору посредством ключевого слова this, которое представляет собой текущий экземпляр класса MDI-контейнера. Затем вызываем метод show о для вывода нового экземпляра дочерней формы. Если нужно вывести более одного дочернего окна, то потребуется повторить две выделенные выше в коде строки для каждого окна. Запустите код, и вы увидите приблизительно то, что показано на рис. 6.1. Это не самый впечатляющий пользовательский интерфейс, однако он вполне подходит в качестве первоосновы.

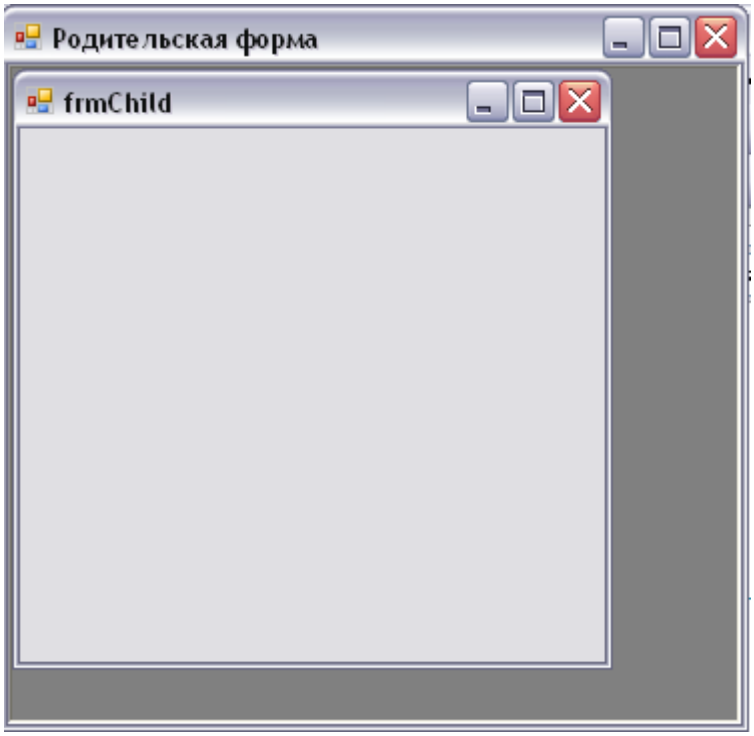

Рис. 6.1. Внешний вид многооконного интерфейса.

2.2. Создание основного меню программы.

Создайте основное меню программы. Добавьте в меню пункты **&Файл** и **&Окно**. Задайте их свойствам name значения MenuItemFile и MenuItemWindow соответственно. Добавьте в меню **&Файл** следующие пункты:

Название	**Текст**
MenuItemNewWindow	&Создать новое окно
toolStripSeparator1	
MenuItemExit	&Выход

Пункт меню **Создать новое окно** будет предназначен для создания дочерних окон, а пункт меню **Выход** для завершения работы приложения.

Пункт меню **&Окно** будет содержать список всех открытых дочерних окон. Такая возможность заложена в меню автоматически. Для этого необходимо присвоить свойству MdiWindowListItem основного меню menuStrip1 имя пункта меню Окно, т. е. menuItemWindow.

Создание дочерних окон должно происходить при выборе пункта меню **Создать новое окно**. Для этого нам необходимо создать обработчик этого пункта меню. Щелкните два раза указателем мыши по имени события Click пункта меню **Создать новое окно** в окне свойств. В код программы добавится обработчик события Click с именем `MenuItemNew_Click`. Добавьте к этому обработчику события представленный ниже код:

```
    private void MenuItemNewWindow_Click(object sender,
EventArgs e)
```

```
{
    // Создание нового экземпляра дочерней формы
        frmChild newChild = new frmChild(this);
        // Вывод созданной формы
        newChild.Show();
            }
```

В функции создается экземпляр класса frmChild с именем newChild. Объект newChild —это дочернее окно, поскольку в конструкторе класса его свойству MdiParent присваивается ссылка на родительское окно. Таким образом, что родительской формой создаваемого окна является главная форма приложения. Все, что остается сделать, это отобразить форму на экране. Для этого используется метод Show.

Добавьте также обработчик пункта меню **Выход**:

```
private void MenuItemExit_Click(object sender, EventArgs
e)
        {
           Application.Exit();
        }
```

Запустите программу. Выберите из меню пункт *Файл/Создать новое окно.* На экране появится дочернее окно с именем frmChild. (рис. 6.2)

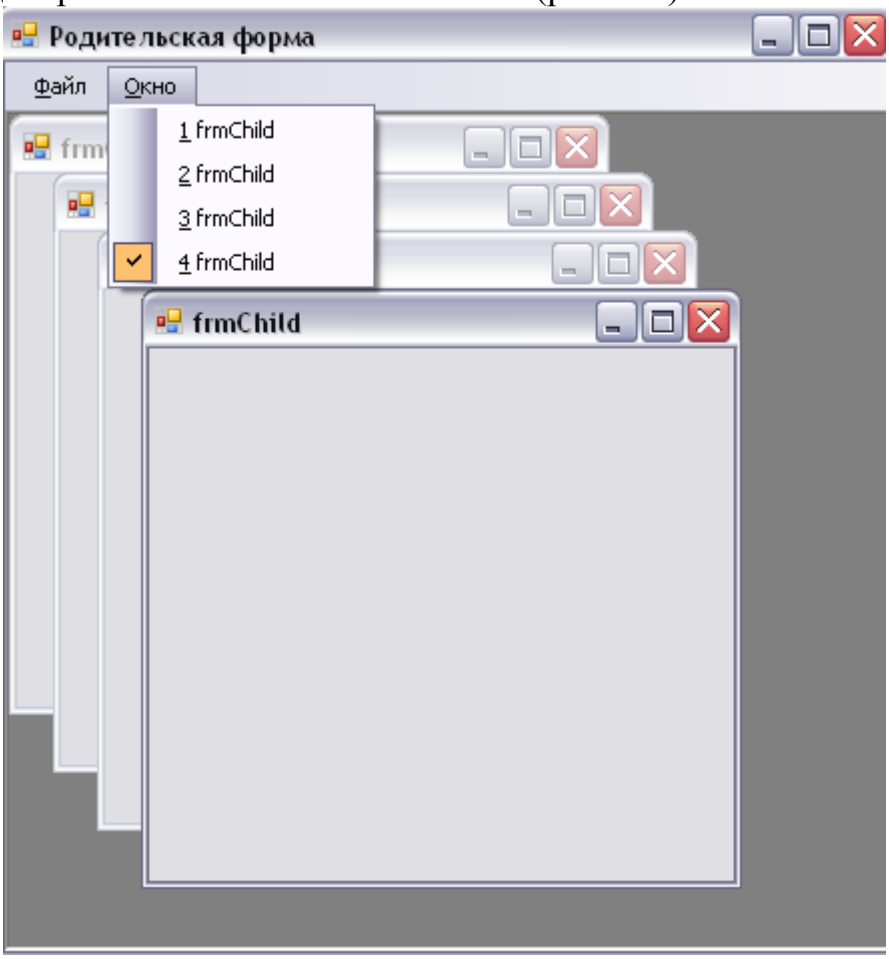

Рис 6.2. Внешний вид готового MDI приложения

2.3. Создание многооконного текстового редактора.

Созданную программу MDIBasic необходимо наполнить

функциональностью. Для этого на дочернюю форму frmChild добавьте один управляющий элемент RichTextBox. Назовите его rtfText и присвойте его свойству Dock значение Fill, для того чтобы окно редактирования заполнило всю форму целиком.

Контекстное меню — это меню, которое открывается, когда пользователь щелкает правой кнопкой мыши на каком-либо объекте, расположенном в форме. Создать контекстное меню можно посредством переноса такого меню в форму и добавления к нему необходимых. Для того чтобы привязать контекстное меню к какому-либо управляющему элементу в форме, следует выбрать соответствующий управляющий элемент и присвоить его свойству ContextMenuStrip указатель на созданное вами контекстное меню.

Создайте контекстное меню для дочерней формы contextMenuStrip1 со следующими пунктами:

Имя	**Текст**
MenuItemBold	Полу&жирный
MenuItemItalic	&Курсив
MenuItemUnderline	Под&черкнутый

Теперь можно перейти к обработке сообщений контекстного меню. Для добавления обработчика событий click выполните двойной щелчок мышью на пунктах контекстного меню и внесите изменения в созданные функции, например, обработчик пункта **Полужирный** должен выглядеть таким образом:

```
    private void MenuItemBold_Click(object sender, EventArgs
e)
            {
                Font          newFont          =          new
Font(rtfText.SelectionFont,   //1
                (rtfText.SelectionFont.Bold ?
    //2
                rtfText.SelectionFont.Style & ~FontStyle.Bold
:   //3
                rtfText.SelectionFont.Style                  |
FontStyle.Bold));    //4
                rtfText.SelectionFont =newFont;
    //5
            }
```

В этой функций создается новый шрифт на основе шрифта, который используется в текущий момент в окне RichTextBox. Если стиль шрифта уже задан, то удаляем его из нового шрифта, в противном случае добавляем требу

емый стиль в шрифт. После этого присваиваем новый шрифт выбранному тексту. В функции используется тернарный оператор сравнения. На строке 2 мы осуществляем проверку, равно ли, значение свойства Bold выбранного шрифта true. Если да, то необходимо сформировать новый стиль для создаваемого шрифта, в котором будут сохранены все имеющиеся у него стили за исключением выделения жирным шрифтом, чем и занимается код, расположенный между символами ? и : на строке 3. Код, расположенный на строке 4 — сразу после двоеточия — будет

выполняться в том случае, если текущее значение свойства Bold равно false, причем на этот раз вместо того, чтобы удалять выделение жирным шрифтом, мы добавляем его. Наконец, вся эта последовательность событий заключается в круглые скобки — этот прием весьма часто используется для отделения логики тернарного оператора от остального кода, хотя в данном случае польза от такого выделения не представляется очевидной.

Аналогично измените код обработчиков пунктов **Курсив** и **Подчеркнутый**, устанавливая соответствующий `FontStyle.`

2.4. Объединение меню родительской и дочерней форм.

Действия, которые может выполнять родительская форма, достаточно ограничены, только команда Создать. Создайте основное меню дочерней формы и добавьте в него следующие элементы:

Название	**Текст**
MenuItemFile	&Файл
MenuItemNew	&Создать
MenuItemOpen	&Открыть
MenuItemSave	&Сохранить

Если запустить приложение, то можно увидеть, что пункт Файл дочернего окна помещен после меню Файл и Окно родительского окна. Причина произошедшего заключается в том, что по умолчанию для пункта пункт Файл дочернего окна установлены свойства:

MergeAction	Append
MergeIndex	-1

Это означает, что данный пункт меню добавляется в конец меню родительского окна. Установите свойство MergeAction в значение MatchOnly, а свойство Visible в значение true. В этом случае пункты меню с одинаковым именем будут объединены. Кроме того, поменяйте свойства для следующих пунктов:

Текст	MergeAction	MergeIndex
&Создать	Insert	1
&Открыть	Insert	2
&Сохранить	Insert	3

Теперь следует добавить код для обработки пунктов меню. Мы постараемся не усложнять методы, соответствующие пунктам Создать, Открыть, Сохранить, и для демонстрационных целей будем использовать фиксированный файл. Для того, чтобы добавить обработчик событий click в какой-либо пункт меню, следует просто дважды щелкнуть на нем мышью. Начнем с двойного щелчка мышью на пункте меню Создать:

```
    private void MenuItemNew_Click(object sender, EventArgs
e)
            {
                this.rtfText.Clear();
            }
```

Параметр sender представляет собой ссылку на пункт меню, на котором произошел щелчок мышью. В данном случае, нам заведомо известно, что таким пунктом является пункт **Создать,** однако обработчик события может быть привязан

к нескольким пунктам, и тогда этот параметр определяет меню, которое следует использовать. Повторите процедуру для пунктов **Открыть, Сохранить**.

```
    private void MenuItemOpen_Click(object sender, EventArgs
e)
            {
                this.rtfText.LoadFile("test.rtf");

            }

    private void MenuItemSave_Click(object sender, EventArgs
e)
            {
                rtfText.SaveFile("test.rtf");
            }
```

Запустите программу.

2.5. Присвоение новому дочернему окну собственного заголовка. Закрытие активного дочернего окна.

Когда происходит щелчок мышью на пункте **Создать новое окно,** создается новое окно. Подобная процедура выполняется в конструкторе. Но при этом возникает проблема. На данный момент все окна обладают одним и тем же заголовком — frmChild, что не позволяет различать их в списке MdiList. Для того чтобы исправить ситуацию, добавим в конструктор формы frmChild еще один параметр, в котором будем передавать текст, предназначенный для вывода в качестве заголовка вновь создаваемого окна:

```
    public frmChild (MDIBasic.frmContainer parent,string
caption)
                {
                    InitializeComponent();
                    //  Присваивание  контейнеру  родителя
данной формы
                    this.MdiParent = parent;
                //Задание заголовка
                this.Text = caption;
               }
```

Это изменение означает, что необходимо изменить вызов конструктора, который создается в конструкторе контейнера:

```
    public frmContainer()
            {
                InitializeComponent();
                // Создание нового экземпляра дочерней формы
                MDIBasic.frmChild      child      =      new
MDIBasic.frmChild(this,"Редактор 1");
                // Вывод созданной формы
                child.Show();
            }
```

Теперь можно создать новые окна с новым заголовком. Измените код обработчика пункта меню **Создать новое окно**:

```
    private void MenuItemNewWindow_Click(object sender,
EventArgs e)
            {       // Создание нового экземпляра дочерней
формы
                    frmChild newChild = new
frmChild(this,"Редактор"+nextFormNumber++);
                    // Вывод созданной формы
                    newChild.Show();
                }
```

Переменная nextFormNumber описывается в самом начале класса следующим образом:

```
    private int nextFormNumber = 2;
```

Мы просто прибавляем единицу к текущему номеру каждый раз, когда открывается новая форма. Причина, по которой номер следующей формы равен 2, а не 1, заключается в том, что для инициализации первого окна в конструкторе используется текст «Редактор 1». После этого мы создаем новый экземпляр редактора и выводим его.

Создайте в пункте меню Окно следующие элементы — Плитка (Tile) и Каскад (Cascade). Существует два возможных способа располагать вновь открываемые документы поверх уже открытых — вертикально и горизонтально; в данном примере мы будем располагать окна горизонтально. Для последовательного расположения нескольких окон с документами имеется только одна возможность. Щелкните два раза мышью на меню Плитка и на меню Каскад и введите следующий код:

```
    private void MenuItemTile_Click(object sender, EventArgs
e)
            {
                this.LayoutMdi(MdiLayout.TileHorizontal);
            }

            private void MenuItemCascade_Click(object
sender, EventArgs e)
            {
                this.LayoutMdi(MdiLayout.Cascade);
            }
```

Метод LayoutMdi основного окна позволяет изменять порядок расположения всех дочерних MDI-окон. Запустите созданный код, поэкспериментируйте с расположением дочерних окон.

Теперь нужно закрыть открытые окна. Добавьте в меню Файл родительской формы пункт **Закрыть**. Форма frmContainer обладает свойством ActiveMdiChild, которое позволяет идентифицировать то дочернее окно, которое мы хотим закрыть. Щелкните два раза мышью на меню **Закрыть** и добавьте следующий код:

```
    private void MenuItemClose_Click(object sender,
EventArgs e)
```

```
            {//  Определение  активного  дочернего  MDI-
окна
        frmChild frm =(frmChild)this.ActiveMdiChild;
        if  (frm  !=  null)  //  Перед  тем  как  использовать
    потолок,
            // необходимо убедиться в том, что он доступен
            {// Закрытие окна
                frm.Close();
            }
            }
```

Сначала мы изменяем тип формы, содержащейся в свойстве ActiveMdiChild на класс `frmChild`. Затем, прежде чем выполнять какие-либо действия с данным экземпляром, мы убеждаемся, что его значение не равно null, а потом вызываем метод Close() для данного окна.

3. Задания.

1. Добавьте в дочернее окно многодокументного тестового редактора панель инструментов с тремя кнопками для изменения стиля шрифта Полужирный, Курсив и Подчеркнутый. Обработчики события click должны быть одни и те же для кнопок панели инструментов и для пунктов контекстного меню.

2. Добавьте в дочернее окно пункт основного меню «Правка» со следующими элементами: «Отменить», «Повторить», «Копировать», «Вырезать», «Вставить». Задать обработчики данных пунктов меню, выполняющие соответствующие действия. В RichTextBox есть методы, точно соответствующие всем пунктам меню, поэтому можно просто использовать функции Undo(), Redo(), Cut(), Copy() и Paste() элемента типа RichTextBox. Предусмотреть блокировку пунктов меню, если их невозможно выполнить, например, если в буфере нет текста для вставки, пункт «Вставить» должен быть заблокирован.

```
this.menuItemCut.Enabled = rtfText.SelectedText.Length > 0 ? true : false;
```