**Задание «Графический интерфейс».** В среде объектно-ориентированного программирования *Microsoft Visual Studio* на языке *Visual C#* разработать графический пользовательский интерфейс, состоящий из двух экранных форм (основной и дочерней), все элементы управления которого полностью настраиваются кодом без использования графического конструктора. Иначе:задача состоит в полной автоматизации компоновки графического пользовательского интерфейса.

По итогам выполнения работы сдаются строго два файла:

- отчёт, выполненный в текстовом редакторе *Microsoft Office Word* (*\*.doc* или *\*.docx*);

- файл с архивом всего решения (*Solution*), написанного на языке *Visual C#* в среде *Microsoft Visual Studio* в режиме *Windows Forms Application* (*\*.zip* или *\*.rar*).

Отправленные поодиночке файлы проверке не подлежат. При отсутствии одного из упомянутых файлов зачёт по заданию не выставляется.

Раздел «Подтверждение соответствия графического пользовательского интерфейса требованиям к оформлению» наполнить демонстрацией скриншотами выполнения каждого из пунктов раздела «Требования к оформлению» из данной постановки задачи.

**Требования к именам файлов:**

**Общий вид формата имени файла:** «*Дата. Задание. Фамилия.mcdx*»

**Формат записи даты:** «*ГГГГММДД*», где *ГГГГ* – четыре цифры текущего года, *ММ* – две цифры текущего месяца, *ДД* – две цифры текущего дня.

**Формат записи задания:** «Задание *NNk*», где *NN* – две цифры номера задания, *k* – обозначение «о», если файл содержит общую часть; обозначение «и», если файл содержит индивидуальную часть; обозначение «ои», если файл содержит как общую, так и индивидуальную части.

**Если устранить замечания по работе удаётся в тот же день:** после фамилии ставится пробел и в круглых скобках записывается номер попытки исправления.

**Примеры правильных имён файлов, которые сдаются на проверку впервые:**

«*20190328. Задание 03и. Иванов.docx*»

«*20190328. Задание 03и. Иванов.rar*»

«*20190328. Задание 03и. Иванов.vsdx*»

**Примеры правильных имён файлов, которые сдаются на проверку повторно в тот же день:**

«*20190328. Задание 03и. Иванов (1).docx*»

«*20190328. Задание 03и. Иванов (1).rar*»

«*20190328. Задание 03и. Иванов (1).vsdx*»

**Внимание!** Не забудьте выполнить автоматическую нумерацию страниц в отчёте.

Отчёт по выполненной работе должен содержать:

0. Титульный лист.

1. Формулировку цели работы.

2. Описание задачи согласно выданному варианту.

3. Составление диаграммы классов, входящих в состав решения.

4. Составление Сети Петри запрограммированного технологического процесса.

5. Составление блок-схем алгоритмов методов в составе решения.

6. Подбор и расчёт тестовых примеров.

7. Листинг кода составленного программного обеспечения.

8. Графический пользовательский интерфейс программного обеспечения и его описание.

9. Подтверждение соответствия графического пользовательского интерфейса требованиям к оформлению.

10. Расчёт тестовых примеров с использованием составленного программного обеспечения.

11. Формулировку вывода о проделанной работе.

Рекомендации к отчёту, доказывающие самостоятельность выполнения работы и упрощающие процедуру проверки отчёта преподавателем:

1. Выполнение дополнительных скриншотов для случаев, когда текстовое описание проделанных действий становится громоздким или трудным к восприятию.

2. Нумерация рисунков (если есть) с подписями, содержащими названия рисунков, например, «Рисунок 1 – Блок-схема алгоритма программы».

**Цель работы (одна из возможных формулировок)**: «Закрепить навыки разработки визуального пользовательского интерфейса, освоить работу с текстовыми файлами в среде *Visual Studio*, научиться взаимно увязывать одни элементы управления с другими, получить представление о перерисовке и перемасштабировании».

**Основное задание на работу**: «В среде объектно-ориентированного программирования *Visual Studio* разработать программу в режиме *Windows Forms Application* на языке *Visual C#*, представляющую собой пользовательский интерфейс, содержащий главное меню, позволяющее:

1. Начать работу с приложением.
2. Прервать работу приложения.
3. Предоставить пользователю справочную информацию о работе с приложением.

Сама программа должна реализовывать вывод в списки значений аргумента и соответствующих им значений функций. Список функций должен обязательно содержать следующие пункты:

1. Логарифм по основанию 2,
2. Тангенс,
3. Возведение в квадрат,
4. Косинус,
5. Натуральный логарифм,
6. Арккосинус,
7. Извлечение корня,
8. Арктангенс,
9. Синус,
10. Десятичный логарифм,
11. Арксинус.

Таблица 1 – Варианты расположения расчётных функций в комбинированном или обычном списке

|  |  |
| --- | --- |
| **№ вар.** | **Порядок индексов функций из списка** |
| **1** | 5, 3, 11, 4, 1, 9, 10, 6, 7, 8, 2. |
| **2** | 4, 3, 8, 7, 11, 2, 10, 9, 6, 1, 5. |
| **3** | 10, 9, 11, 8, 6, 5, 7, 4, 3, 2, 1. |
| **4** | 11, 2, 9, 4, 6, 5, 7, 8, 3, 10, 1. |
| **5** | 7, 3, 8, 4, 6, 2, 11, 9, 10, 5, 1. |
| **6** | 10, 9, 5, 1, 3, 11, 2, 6, 7, 8, 4. |
| **7** | 1, 7, 3, 9, 5, 11, 10, 4, 2, 6, 8. |
| **8** | 4, 3, 7, 5, 6, 2, 8, 9, 11, 10, 1. |
| **9** | 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11. |
| **10** | 7, 5, 1, 4, 9, 8, 10, 3, 11, 6, 2. |
| **11** | 8, 6, 11, 2, 9, 10, 3, 4, 5, 1, 7. |
| **12** | 2, 1, 10, 11, 5, 9, 8, 6, 7, 3, 4. |
| **13** | 11, 8, 7, 5, 9, 10, 2, 4, 1, 6, 3. |
| **14** | 8, 2, 10, 7, 3, 11, 5, 9, 6, 1, 4. |
| **15** | 9, 1, 6, 7, 4, 11, 5, 10, 8, 3, 2. |
| **16** | 5, 4, 8, 2, 3, 1, 10, 9, 6, 11, 7. |
| **17** | 11, 7, 10, 6, 9, 3, 4, 8, 5, 2, 1. |
| **18** | 1, 7, 2, 6, 5, 9, 10, 3, 11, 4, 8. |
| **19** | 6, 7, 11, 8, 9, 3, 5, 4, 10, 2, 1. |
| **20** | 3, 2, 1, 11, 6, 9, 7, 8, 4, 10, 5. |

**Требования к оформлению**:

1. Заголовок формы должен содержать надпись вида: «Задание №41 выполнил: [Фамилия И.О. автора]; Номер варианта: [Номер]; Дата выполнения: [дд/мм/гггг].

2. Дата выполнения проставляется в момент, когда программа считается законченной и по ней можно готовить итоговый отчёт о выполнении работы.

3. Чётные варианты отключают стандартный блок управления формой и создают авторские кнопки «Свернуть», «Развернуть», «Закрыть» внизу формы.

4. Нечётные варианты заменяют стандартный курсор формы со «стрелки» на «руку».

5. Справочная информация должна быть вызвана в дочерней форме и считана в статический по размеру ярлык из текстового файла (чётные варианты) или в статическое по размеру текстовое поле с выставленным запретом на редактирование текстовой информации (нечётные варианты).

6. В режиме конструктора форма не должна содержать ни одного компонента.

7. Решение должно содержать только одну единственную форму. Форма со справкой настраивается программным кодом.

8. Компоненты формы со справкой должны быть чувствительными к изменению размера формы по ширине и высоте (обязательно подключение события *Resize*).

9. Названия функций должны быть элементами комбинированного списка (чётные варианты) или обыкновенного списка (нечётные варианты).

10. Выбор конкретной функции из списка инициирует заполнение некоторого файла значениями аргумента и значениями функции. Разделитель данных подбирается согласно таблице вариантов.

Таблица 2 – Варианты разделителей, записываемых в файл

|  |  |
| --- | --- |
| 1. | $ |
| 2. | % |
| 3. | &\* |
| 4. | <> |
| 5. | { |
| 6. | $#@ |
| 7. | ^^ |
| 8. | ( |
| 9. | . |
| 10. | >> |
| 11. | , |
| 12. | ; |
| 13. | : |
| 14. | ++ |
| 15. | = |
| 16. | !@ |
| 17. | ||| |
| 18. | пробел |
| 19. | \_ |
| 20. | … |

11. Нечётные варианты заменяют цвет подложки в соответствии с представленной ниже таблицей.

Таблица 3 – Варианты цветового оформления подложки

|  |  |
| --- | --- |
| 1. | *ActiveCaption* |
| 3. | *AppWorkspace* |
| 5. | *ControlText* |
| 7. | *Desktop* |
| 9. | *GradientActiveCaption* |
| 11. | *Highlight* |
| 13. | *HotTrack* |
| 15. | *Info* |
| 17. | *MenuHighlight* |
| 19. | *Window* |

Подобрать читаемый цвет текста к предложенной по варианту схеме.

12. Предусмотреть кнопку, нажатие на которую инициирует заполнение двух списков (один список под значения аргумента, другой список – под значения функции, округлённых по правилу *(N mod 5) + 1* для знаков после плавающей запятой, где *N* – номер варианта) данными о функциональной зависимости из файла. Обучающиеся с чётными номерами вариантов обновляют существующие основные списки, с нечётными номерами вариантов выводят значения, считанные из файла, в дополнительные списки.

13. В качестве исходного состояния функциональных компонентов главной формы принимается либо недоступное состояние (*.Enabled = false*) – чётные варианты, либо невидимое состояние (*.Visible = false*) – нечётные варианты.

14. Кнопка «Начало работы с приложением» должна реализовывать активацию доступа пользователя к функциональным компонентам или отображение функциональных компонентов на форме для пользователя.

15. Все элементы программы должны носить значащие имена переменных, в которых отражено существо этих элементов, например, форма – *frmMain*, ярлык – *lblHelp*, комбинированный список – *cmbFigures* и т.д.

16. Выбор значения абсциссы в списке должен инициировать выбор соответствующего ему значения ординаты и наоборот – выбор значения ординаты должен инициировать выбор соответствующего ему значения абсциссы (работа с параметрами *SelectedIndex* или *SelectedItem* списков).

**Варианты индивидуального задания:**

Таблица 4 – Варианты дополнительных функциональных зависимостей, рассчитываемых программой

|  |  |  |  |
| --- | --- | --- | --- |
| **Функция** | **Поз.** | **Функция** | **Поз.** |
| 1. 1 / log2x | 3 | 13. (x + 1) / ((x2 - 3·x)·(x - 0,3)) | 7 |
| 2. (x2 - x + 7) / (x + 1) | 1 | 14. |log2 |x| - 1| | 10 |
| 3. ((1 – x) / (1 + x))4/5 | 8 | 15. (3√x2) / (x2 - 1) | 3 |
| 4. 2 – (1 / (2x – 1)) | 4 | 16. x / (x2 - 5) | 2 |
| 5. 1 / cos(x) | 2 | 17. log2((x + 1) / (x - 1)) | 4 |
| 6. 1 / sin(x) | 5 | 18. log2 x / (log2 x - 3) | 8 |
| 7. 2·x – (1 / (2·x – 3)) | 11 | 19. |x| / (1 + x) | 9 |
| 8. log-21/3 3· (x2+2·x) | 9 | 20. 1 / tg(x) | 2 |
| 9. 1 / (2·x – 3) | 6 | 21. x / ((x2 - 5·x + 6)·(x + 0,3)) | 11 |
| 10. log1/2 (x2+2·x) | 8 | 22. x / sin(x) | 7 |
| 11. x / cos(x) | 10 | 23. x / (√(x2 - 5) | 3 |
| 12. ln(x) / x | 4 | 24. 1 / (√(x2 - 1) | 1 |

**Памятка программисту (автоматизация компоновки меню):**

1. Необходимо заранее продумать структуру меню, например:

Таблица 5 – Структура разрабатываемого меню

|  |  |  |
| --- | --- | --- |
| Файл |  | Справка |
| Старт > | > Начало работы | Показать справку |
|  | > Сброс |  |
| Выйти |  |  |

2. Продумать значащие имена пунктов меню:

Таблица 6 – Значащие имена пунктов меню

|  |  |  |
| --- | --- | --- |
| *mnuFile* |  | *mnuHelp* |
| *mnuStart* | *mnuStartRun* | *mnuHelpShow* |
|  | *mnuStartReset* |  |
| *mnuExit* |  |  |

3. Объявление переменных под пункты меню прописываются перед конструктором главной формы проекта для обеспечения возможности обращения к ним из всех методов создаваемой программы.

4. За полосу меню отвечает класс *MenuStrip* (*namespace System.Windows.Forms*).

5. За подпункт меню отвечает класс *ToolStripMenuItem* (*namespace System.Windows.Forms*).

Далее следует пример реализации приложения с автособираемыми при компиляции проекта элементами управления. Визуально создаваемый проект до момента запуска у всех обучающихся должен выглядеть идентично и по аналогии с изображённым на Рисунке 1 представлением.

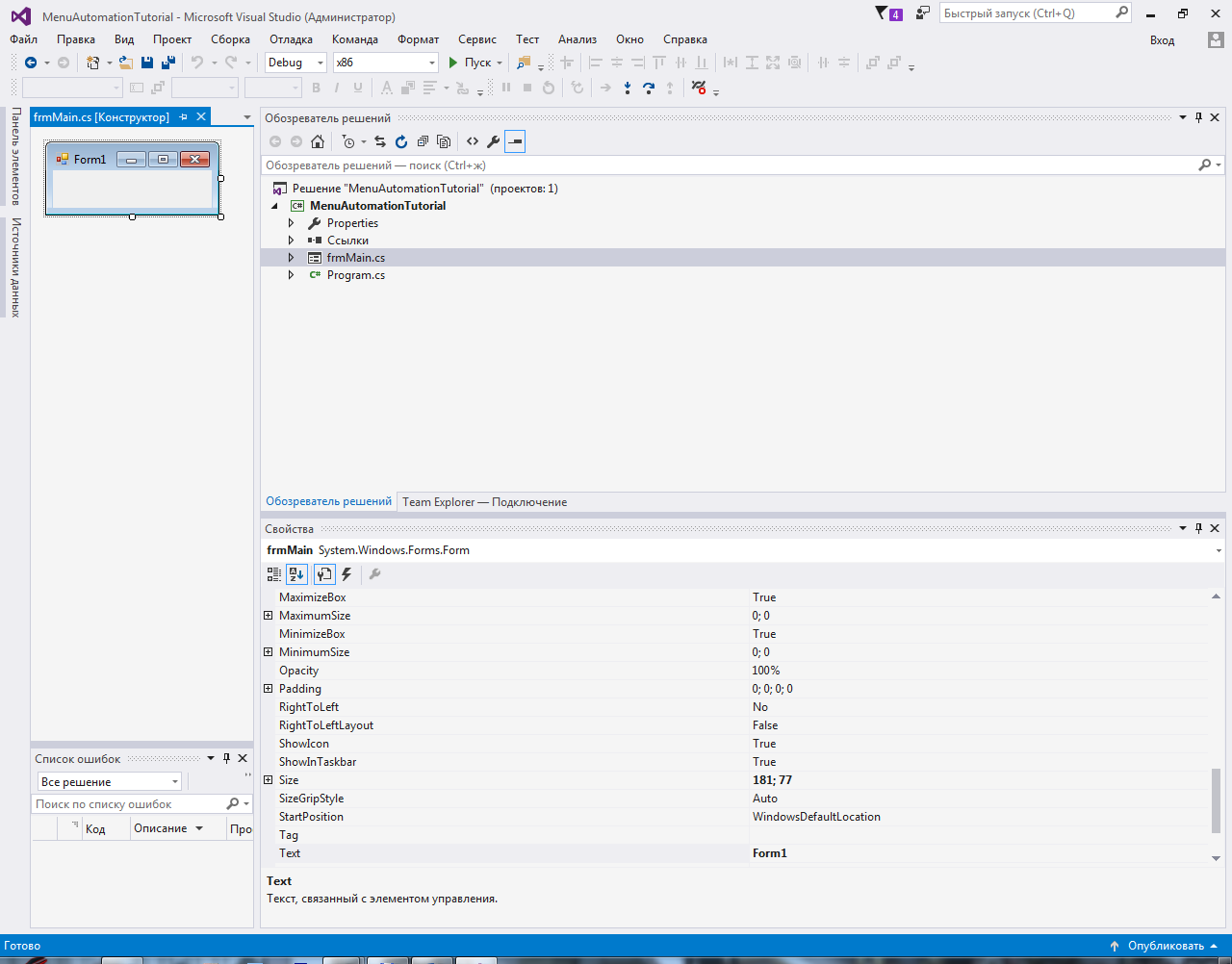


Рисунок 1 – Графический пользовательский интерфейс и наполнение проекта в обозревателе решений

Все необходимые обработчики событий подключаются к проекту по загрузке главной формы проекта. Все подключения обработчиков событий проводятся по аналогии с примером, представленным на Рисунке 2.

Записывается имя элемента управления, который должен отрабатывать интересующее событие (в примере имя опущено [избыточна запись *this.Resize*, хотя она и способствует лучшему восприятию написанного кода] и сразу записано подключаемое событие, поскольку оно относится к самой главной форме), ставится точка и записывается интересующее событие. В выпадающем перечне все события обозначаются пиктограммой в виде молнии. Далее записывается унарный оператор добавления / увеличения «+=». После постановки оператора «+=» среда предлагает посредством нажатия на табуляцию «*Tab*» автоматически создать обработчик события.

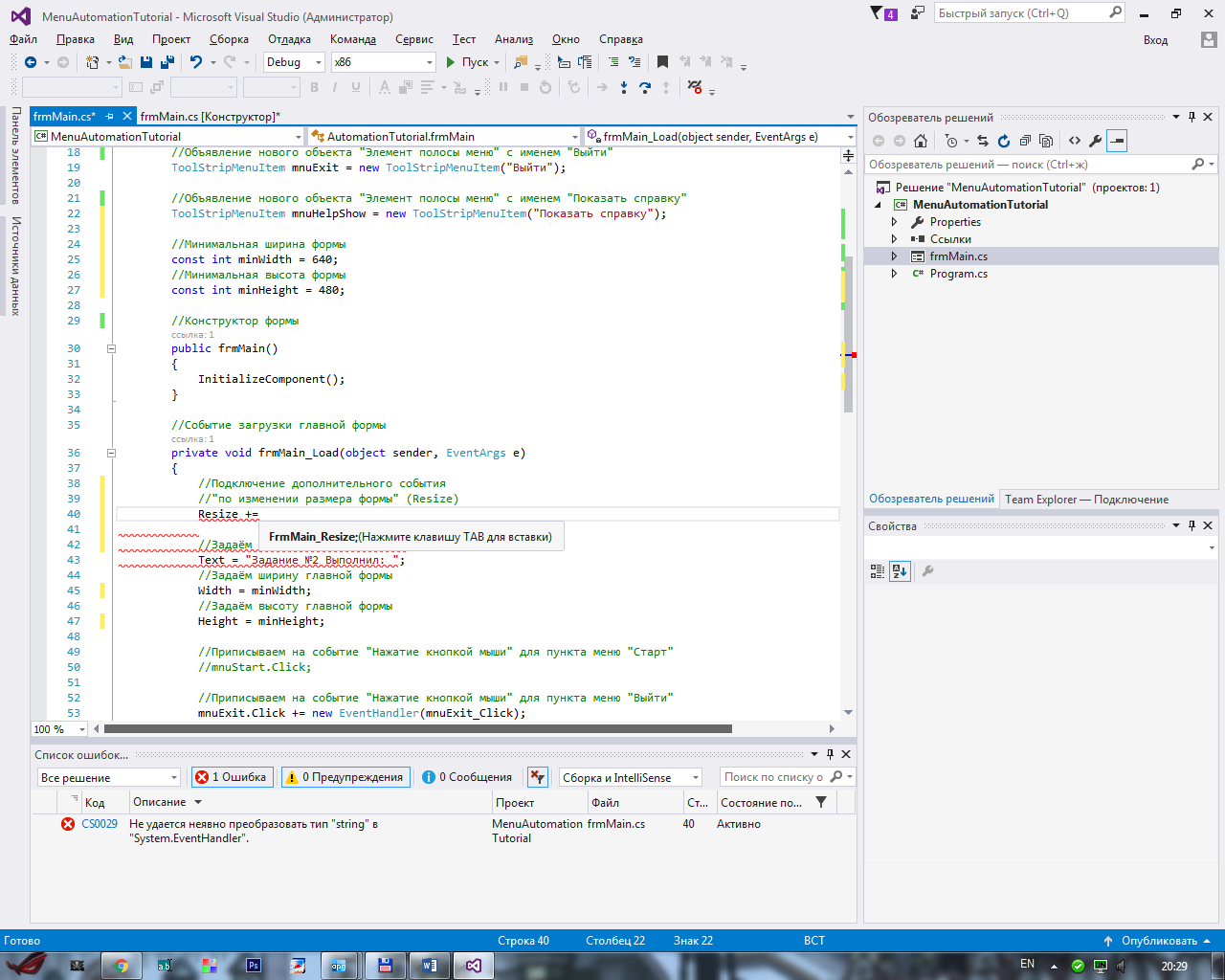


Рисунок 2 – Пример автоматической генерации обработчика события *Resize()*, происходящего при изменении размеров элемента управления

После нажатия на «*Tab*» генерируется шаблон обработчика событий, как показано на Рисунке 3.

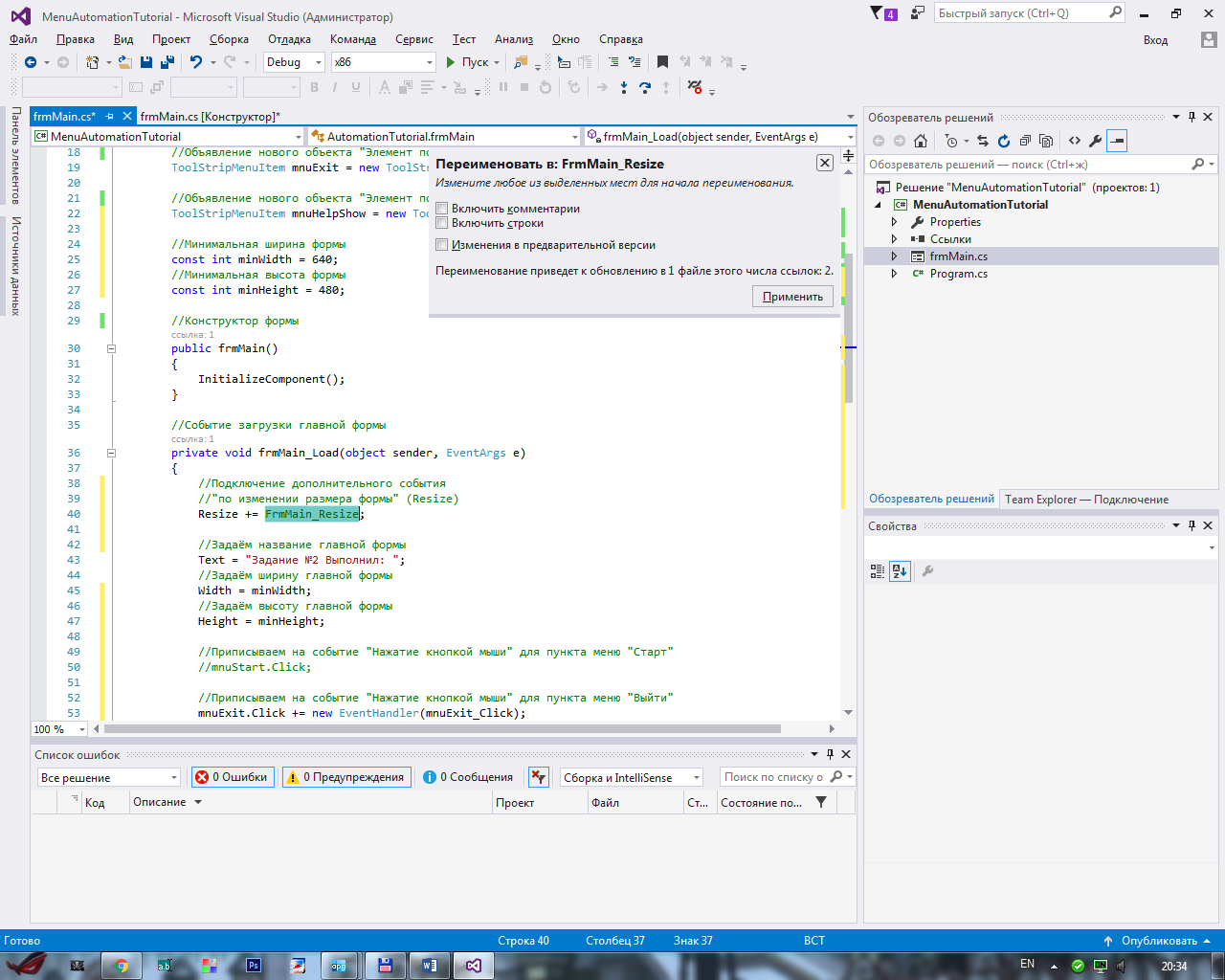


Рисунок 3 – Событие *Resize()* по запросу автоматически связалось с обработчиком события *FrmMain\_Resize*

Повторное нажатие на «*Tab*» переводит курсор к месту расположения созданного шаблона для обработки события (Рисунок 4).

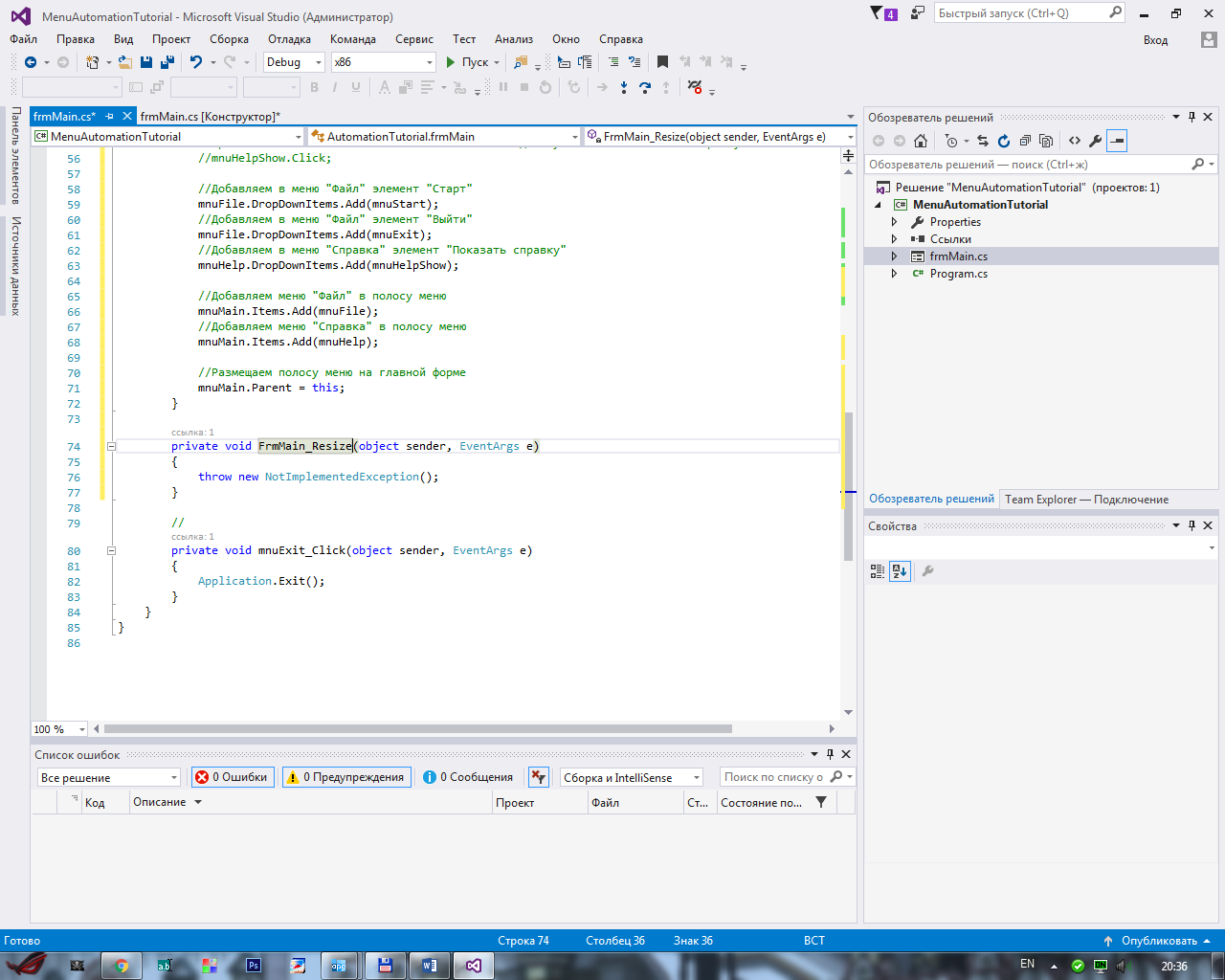


Рисунок 4 – Автоматически созданный шаблон обработчика событий

При изменении размеров экранной формы часто важно не допускать сокращения ширины и высоты ниже стандартных, заранее известных значений во избежание наслаивания элементов управления друг на друга, потому прибегают к принудительному выставлению известных размеров при попытках пользователя выйти за установленные границы. Потому вместо стандартной конструкции обработчика исключений, записанной в теле обработчика событий, следует записать отмеченную выше обработку. Соответствующая запись представлена на Рисунке 5.

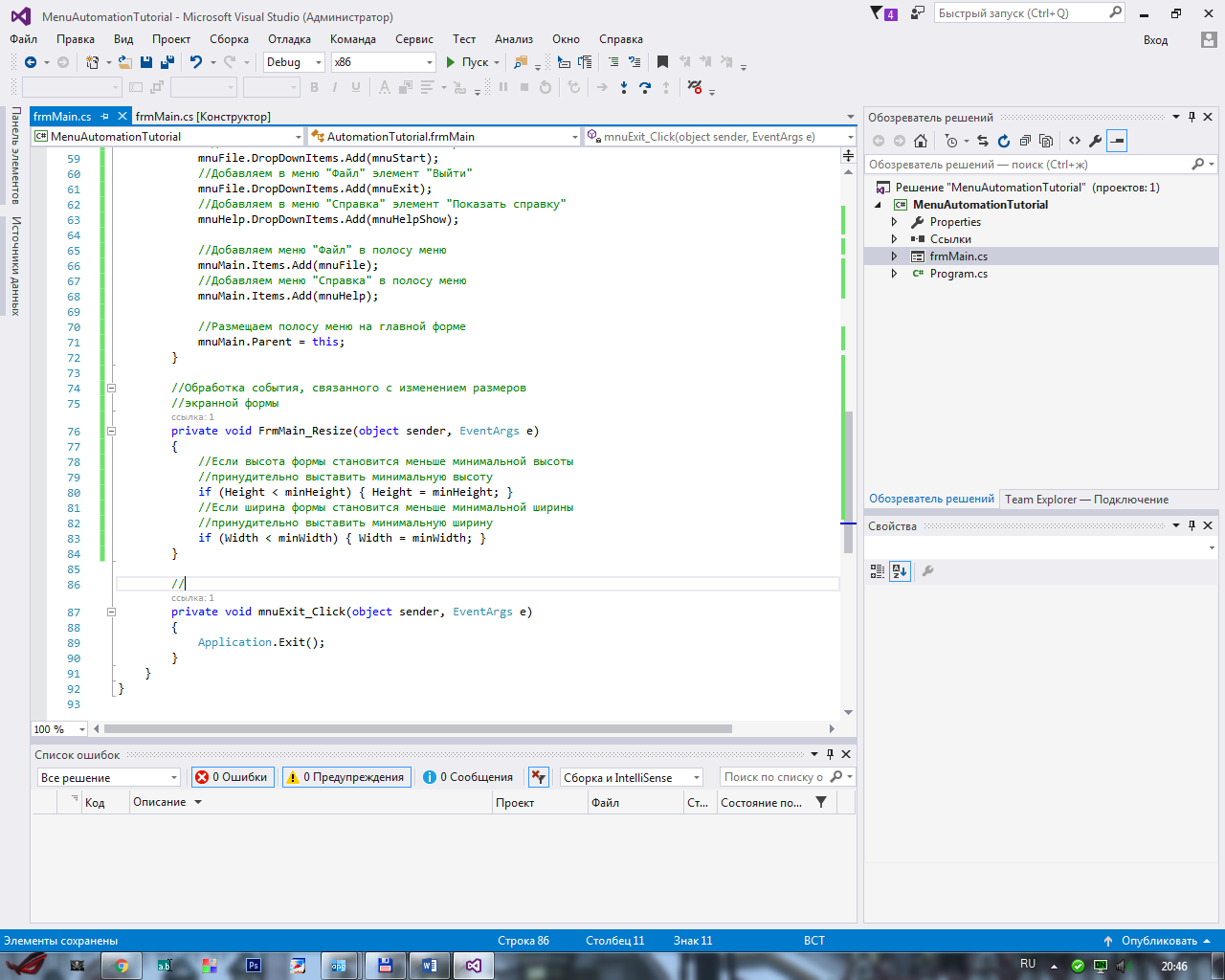


Рисунок 5 – Пример обработки, запрещающей пользователю устанавливать размеры формы меньше значений, заданных по умолчанию

Далее на Рисунках 6 и 7 приведены примеры кода, необходимого для создания меню, рассмотренной в Таблице 5 структуры.

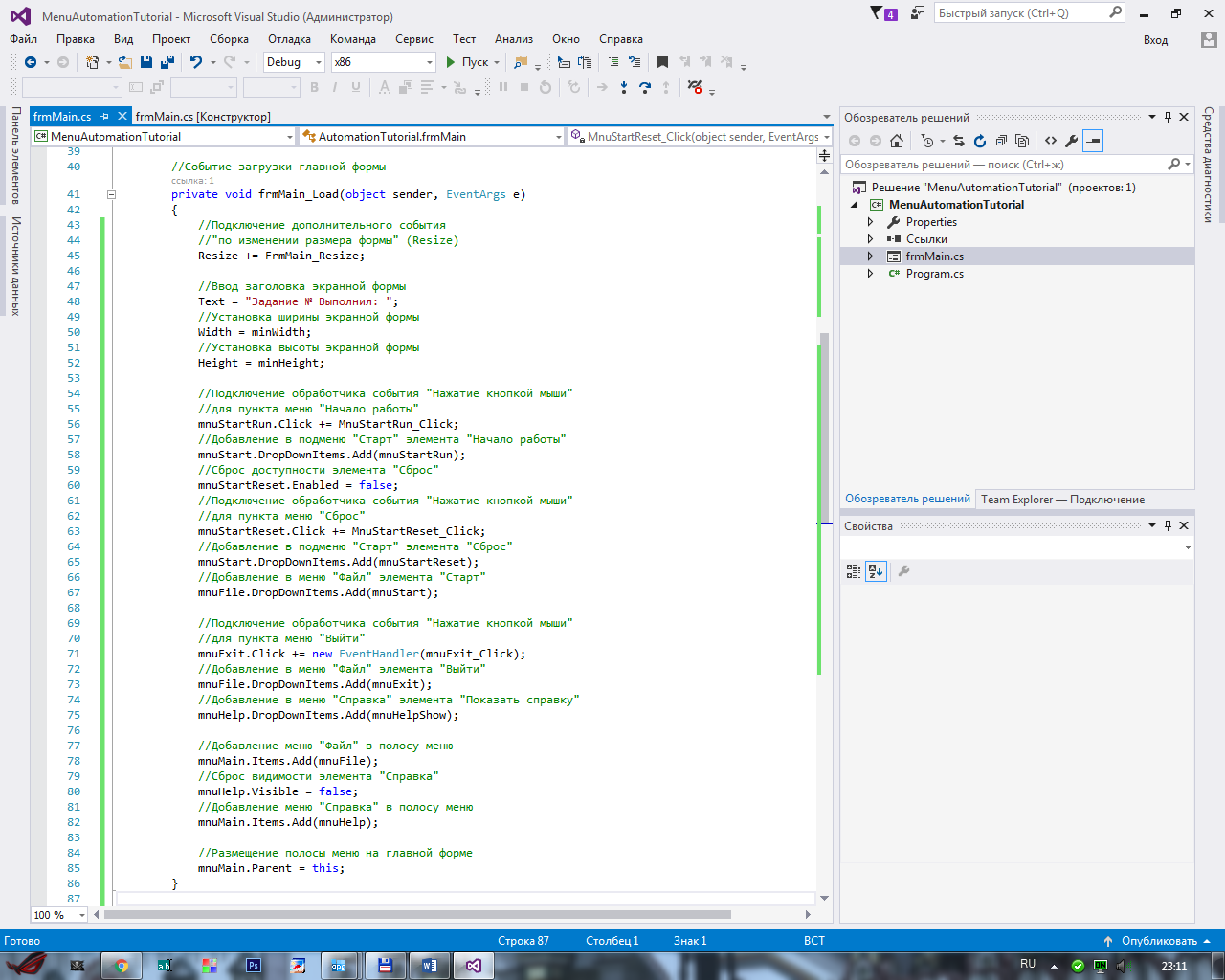


Рисунок 6 – Кодовая настройка главного меню главной формы проекта

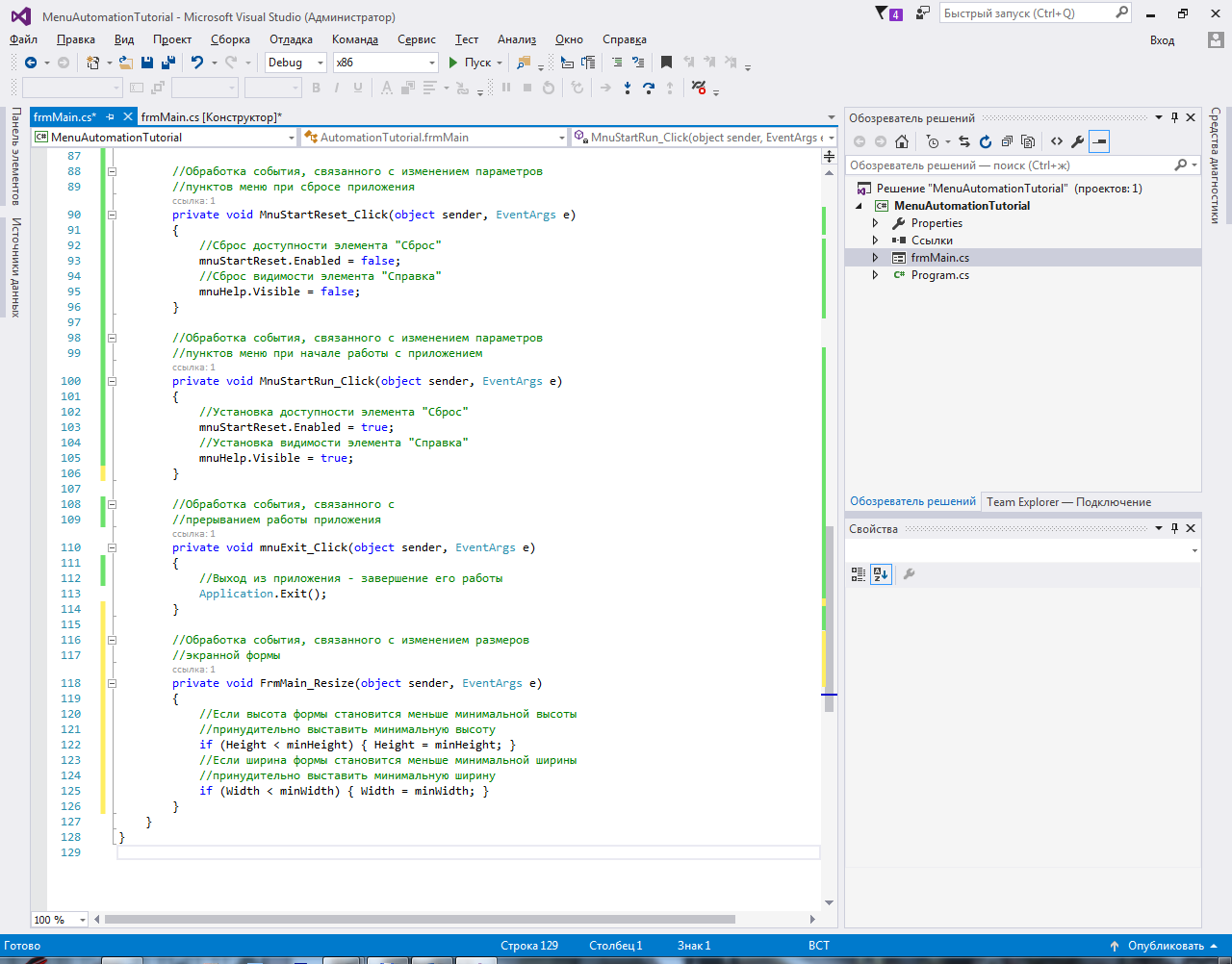


Рисунок 7 – Содержимое обработчиков событий, связанных с пунктами меню главной формы

На Рисунке 8 показано исходное состояние меню до выдачи инструкции к началу работы с приложением.

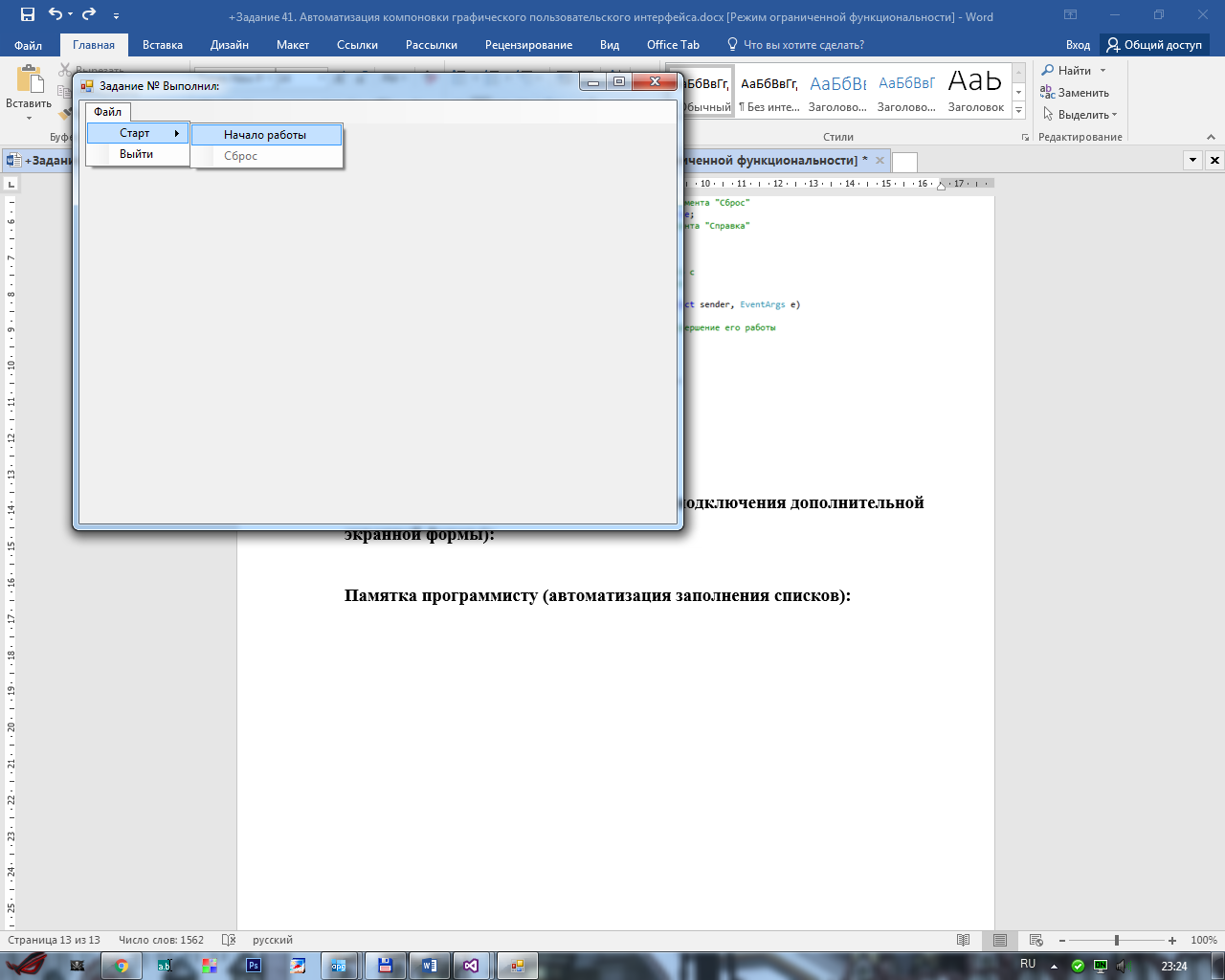


Рисунок 8 – Исходное состояние запущенного на исполнение проекта

На Рисунке 9 показано состояние меню после выдачи инструкции к началу работы с приложением (становится доступным пункт «Сброс» и видимым пункт «Справка»).

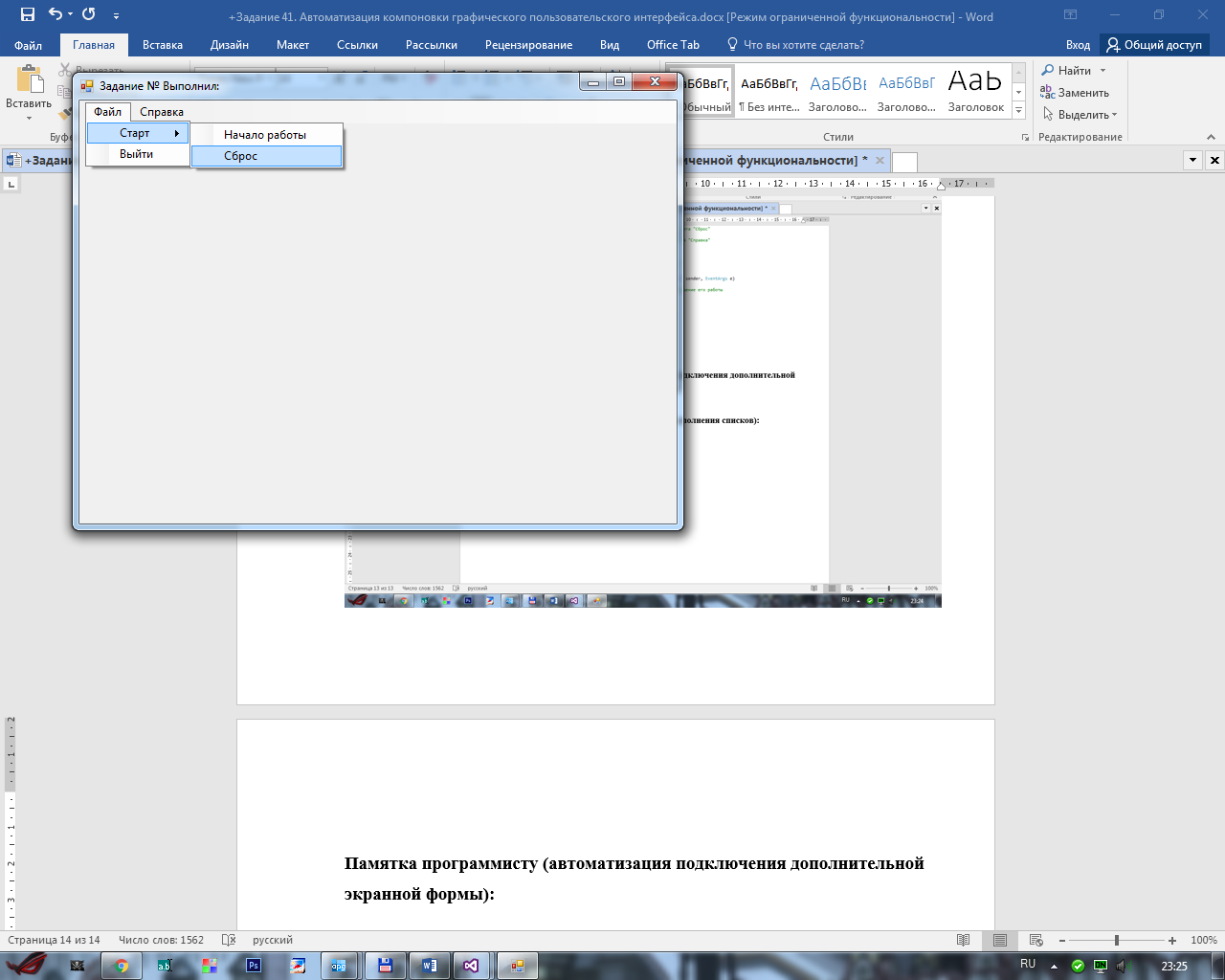


Рисунок 9 – Рабочее состояние главной формы проекта

**Памятка программисту (автоматизация подключения дополнительной экранной формы):**

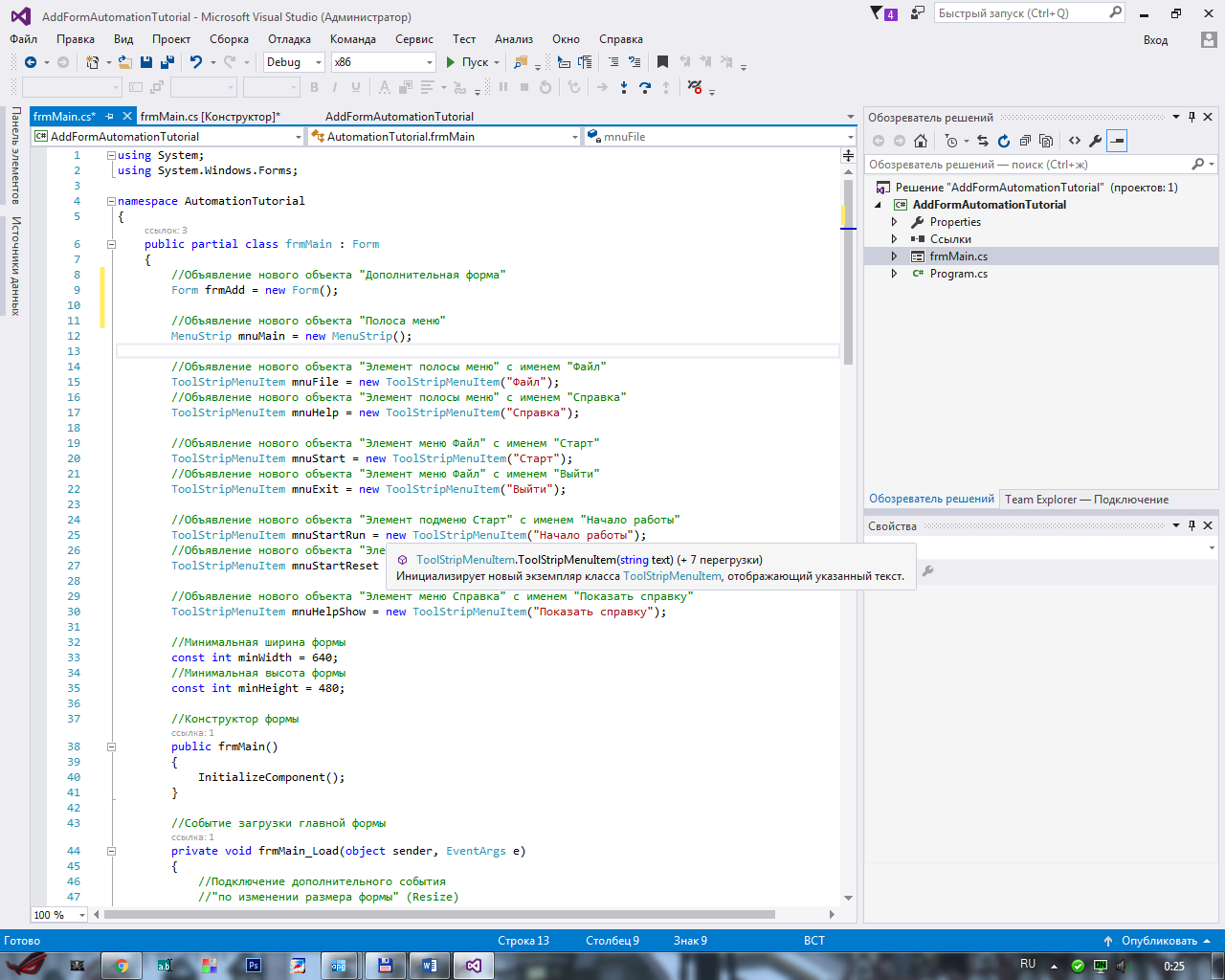


Рисунок 10 – Пример создания объекта дополнительной экранной формы общего вида

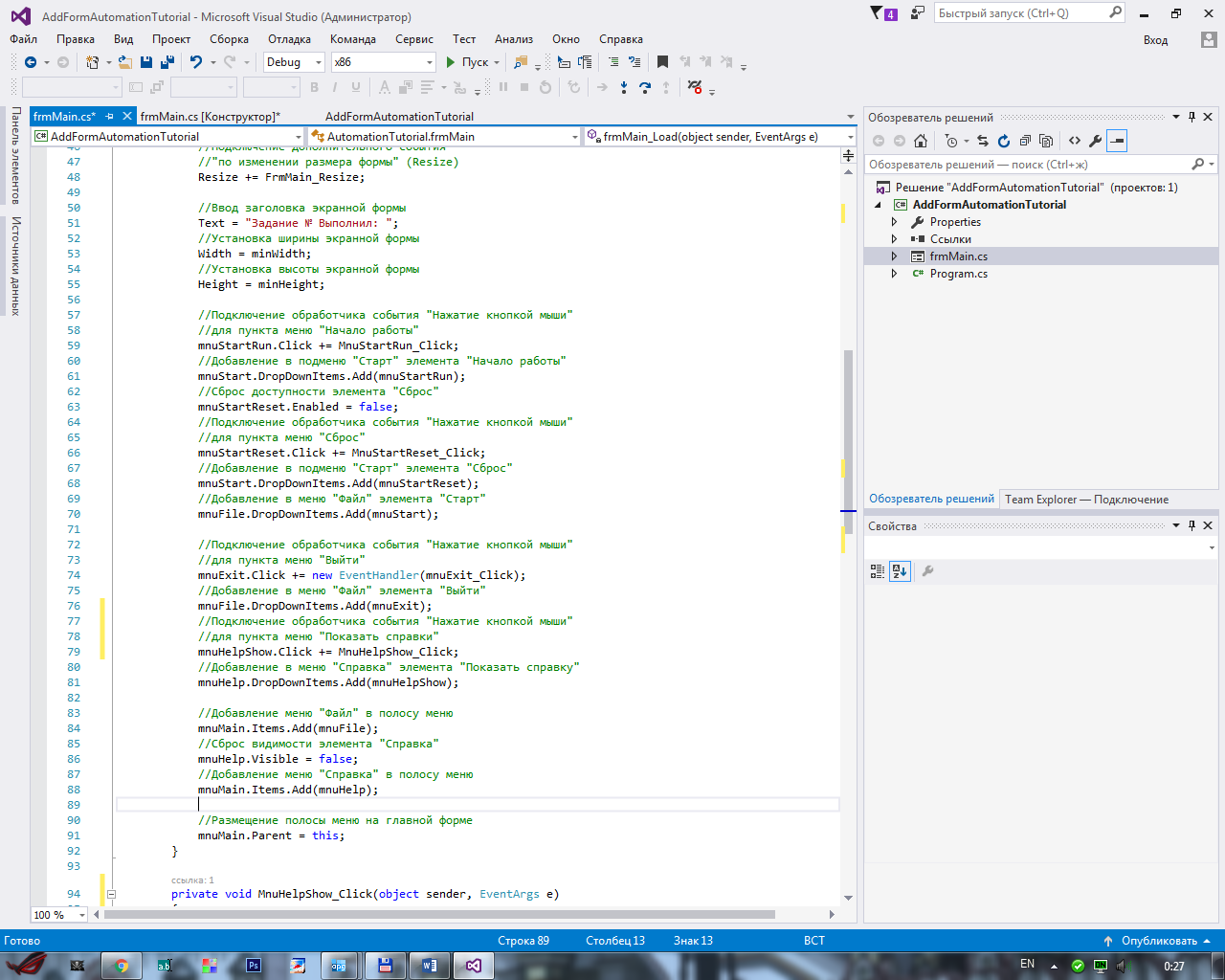


Рисунок 11 – Привязка обработчика события для вызова дополнительной экранной формы (*MnuHelpShow\_Click*)

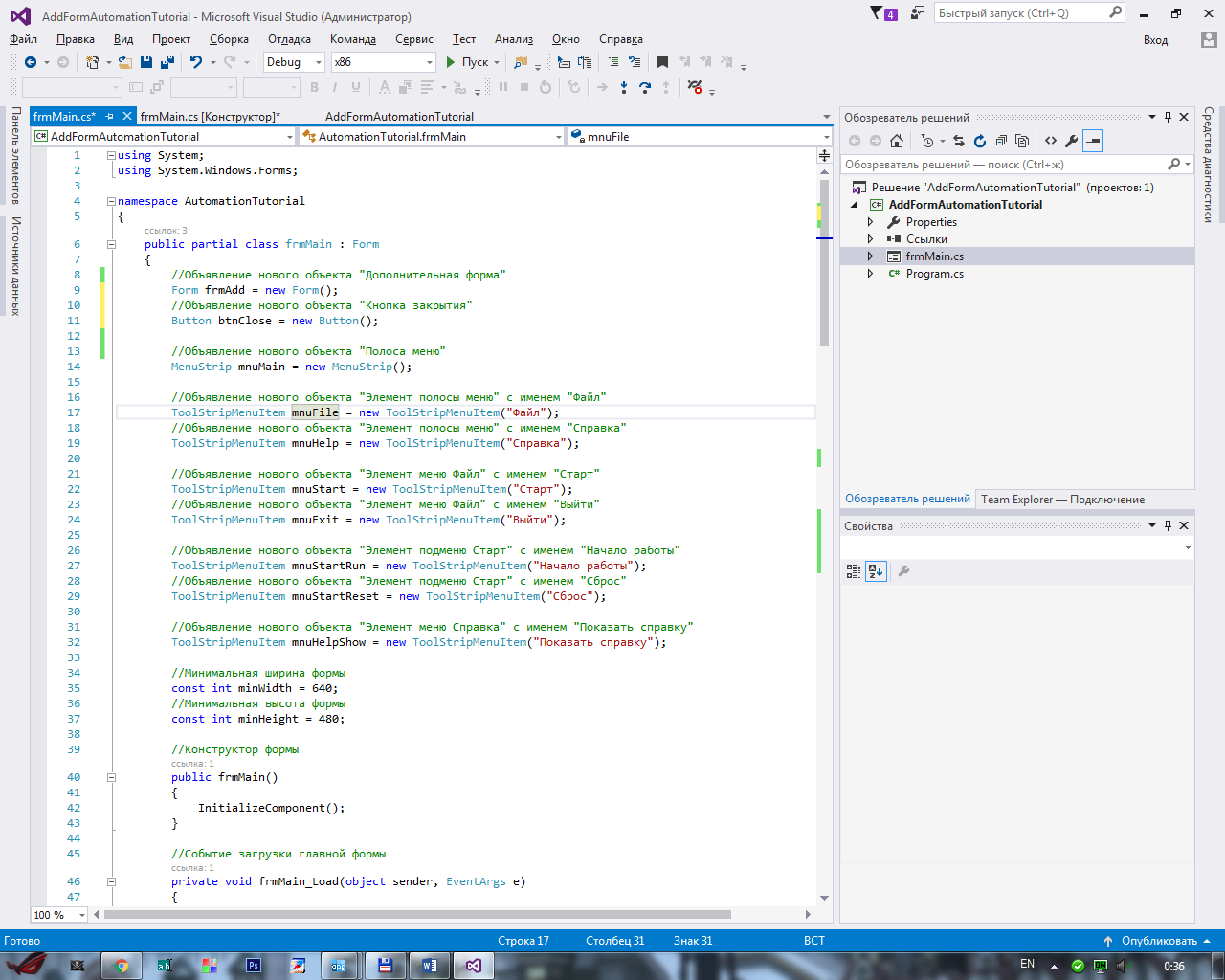


Рисунок 12 – Пример создания объекта типа «Кнопка»

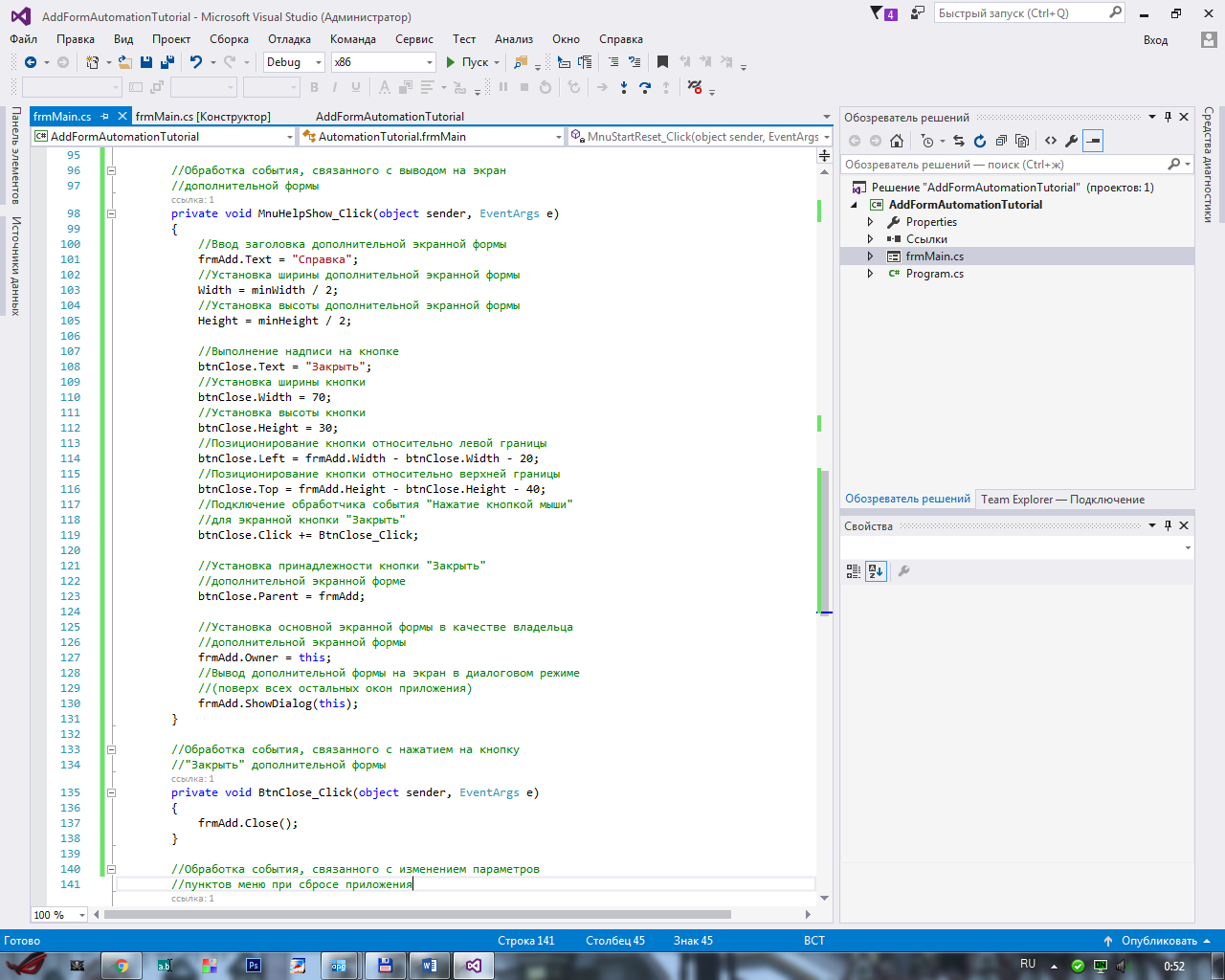


Рисунок 13 – Текст кода обработчика события, связанного с нажатием на пункт меню «Показать справку» и обработчика события, связанного с нажатием на «Кнопку»

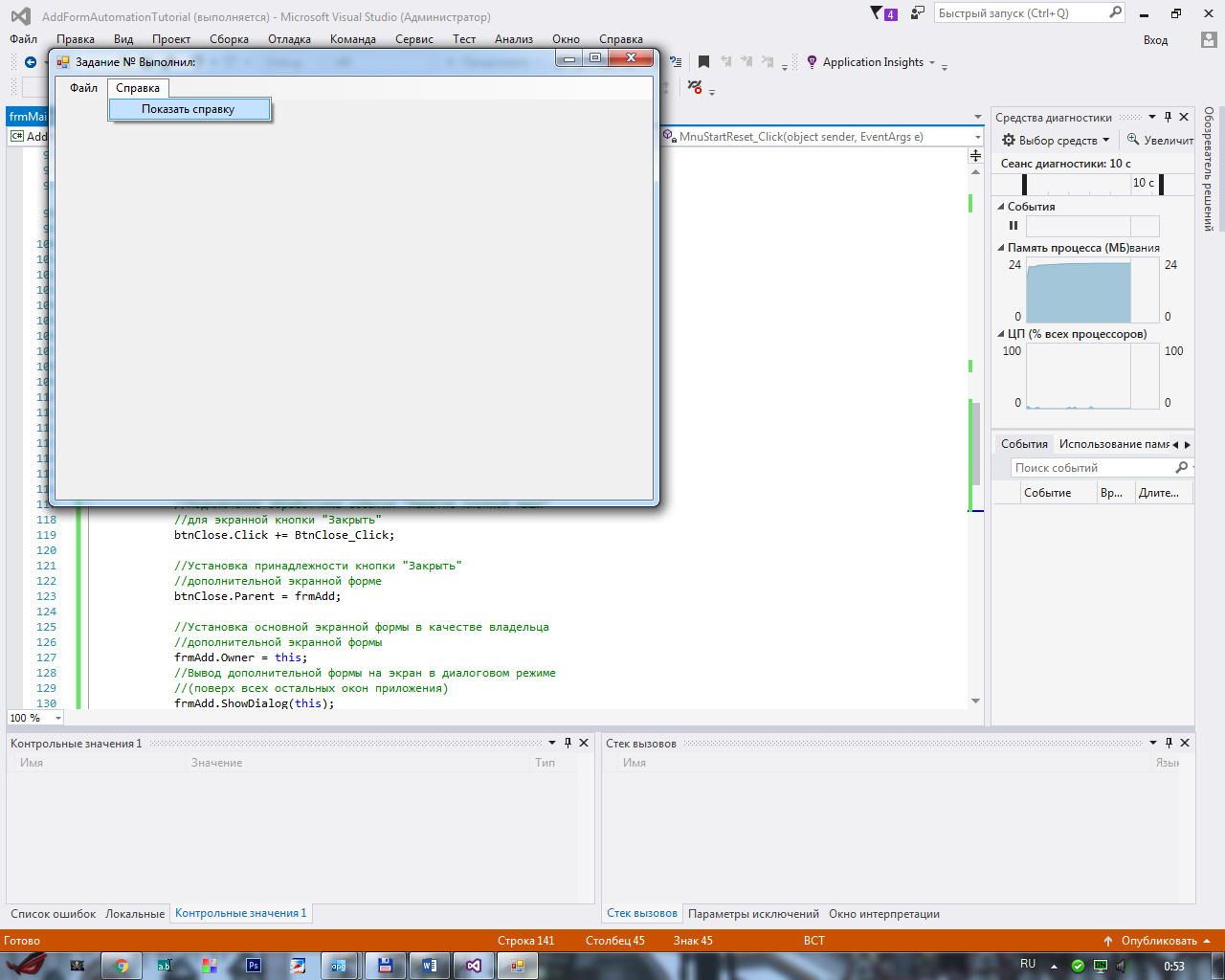


Рисунок 14 – Переход к отображению справки

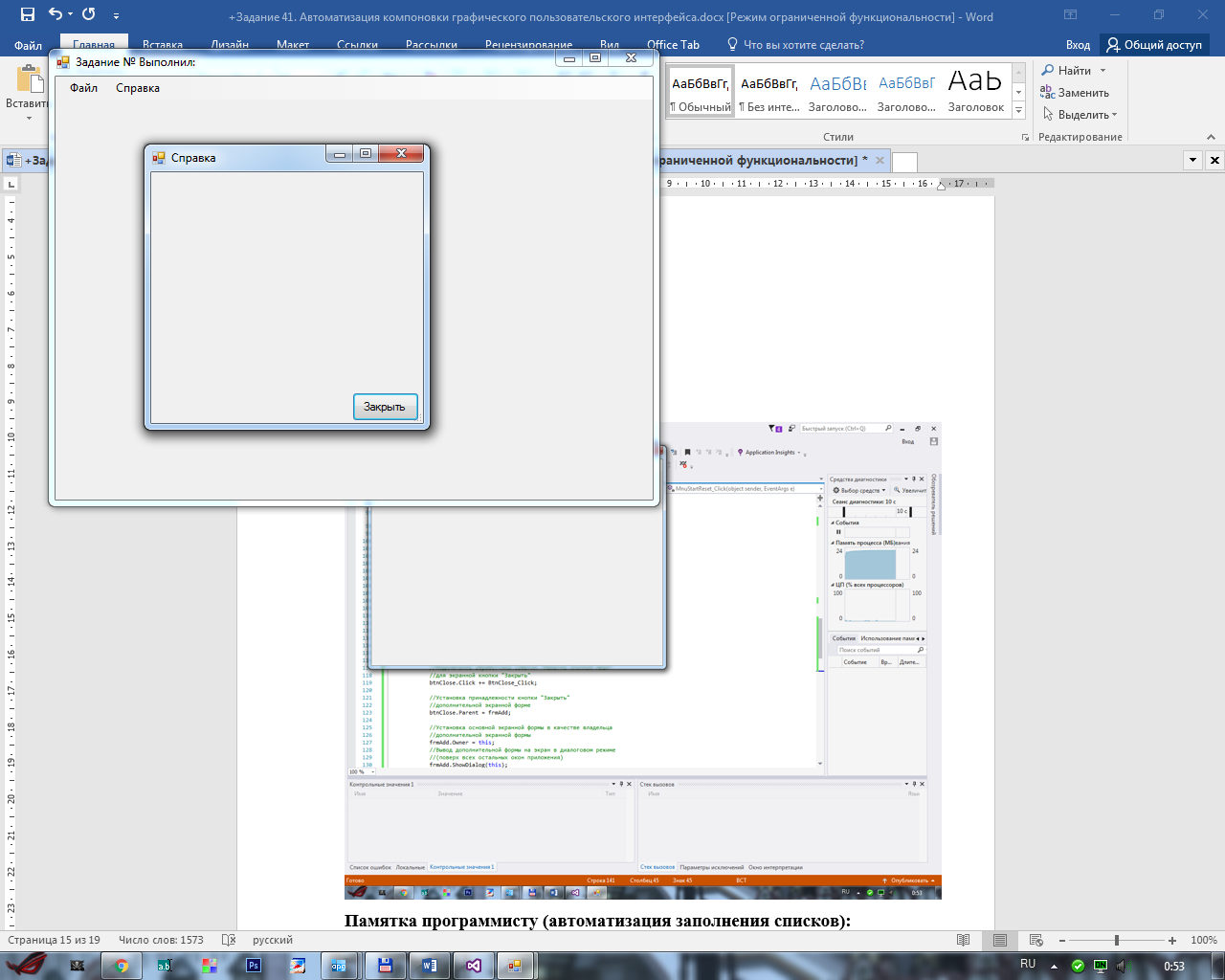


Рисунок 15 – Как результат вызвана дополнительная экранная форма, равная половине основной, с размещённой на ней кнопкой для закрытия

**Памятка программисту (автоматизация заполнения списков):**

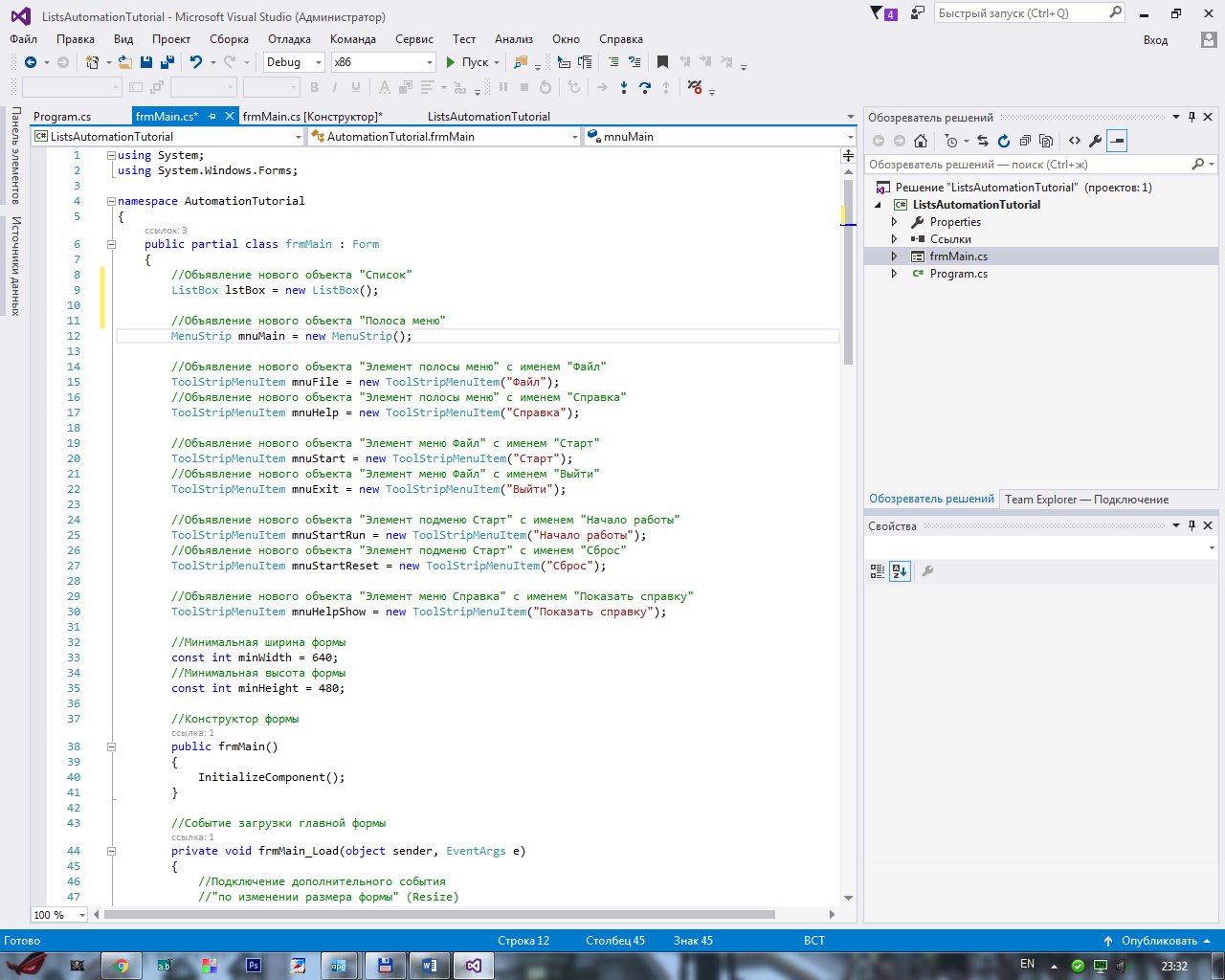


Рисунок 16 – Создание объекта типа «Список»

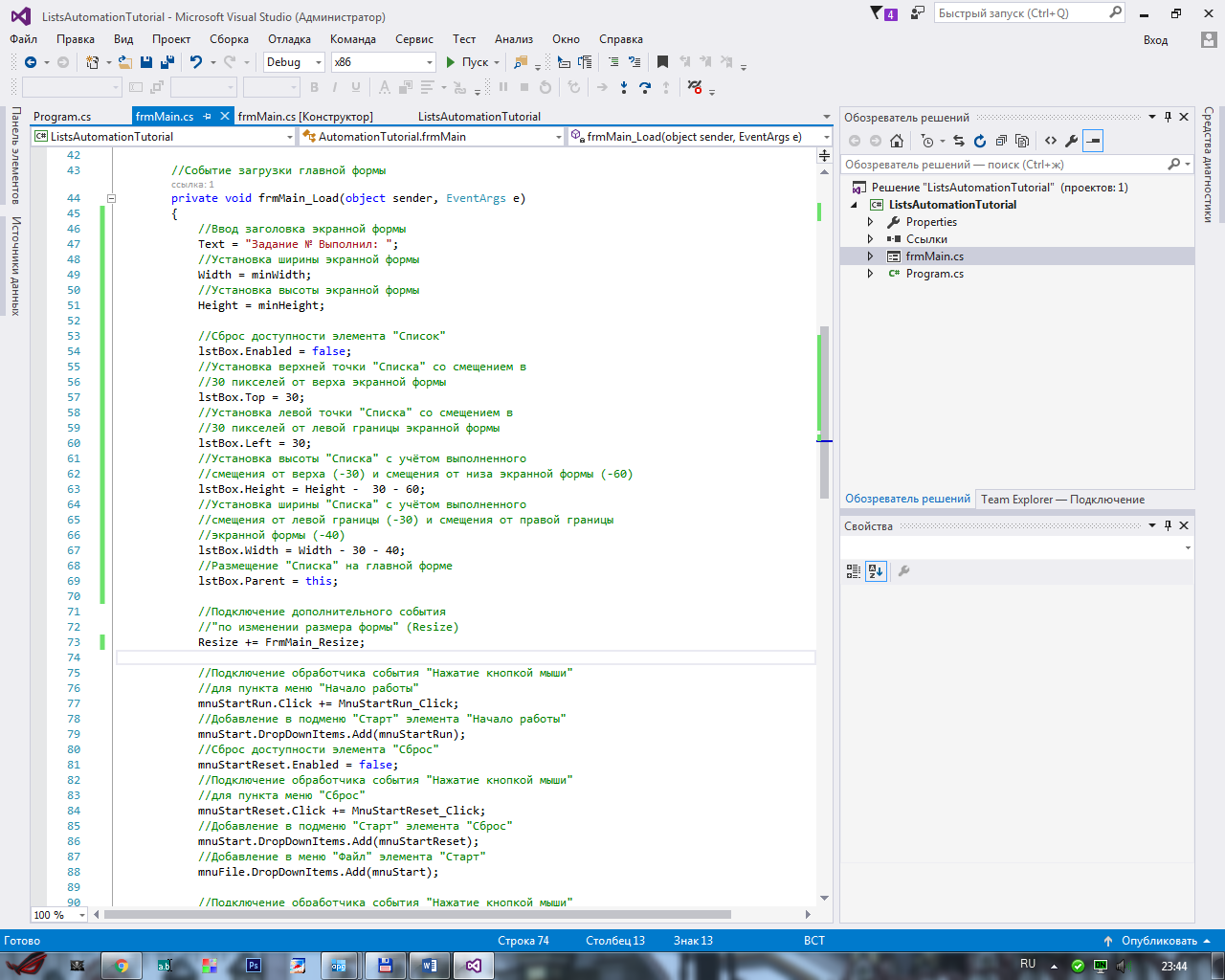


Рисунок 17 – Текст кода для размещения списка на главной форме с характерными отступами от её границ

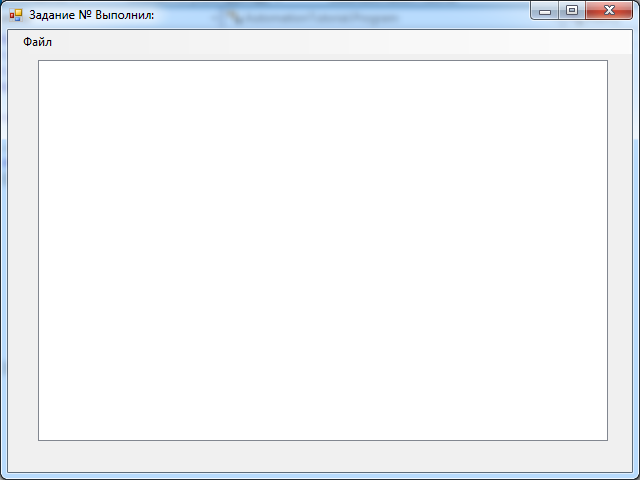


Рисунок 18 – Результат вывода списка на главную форму

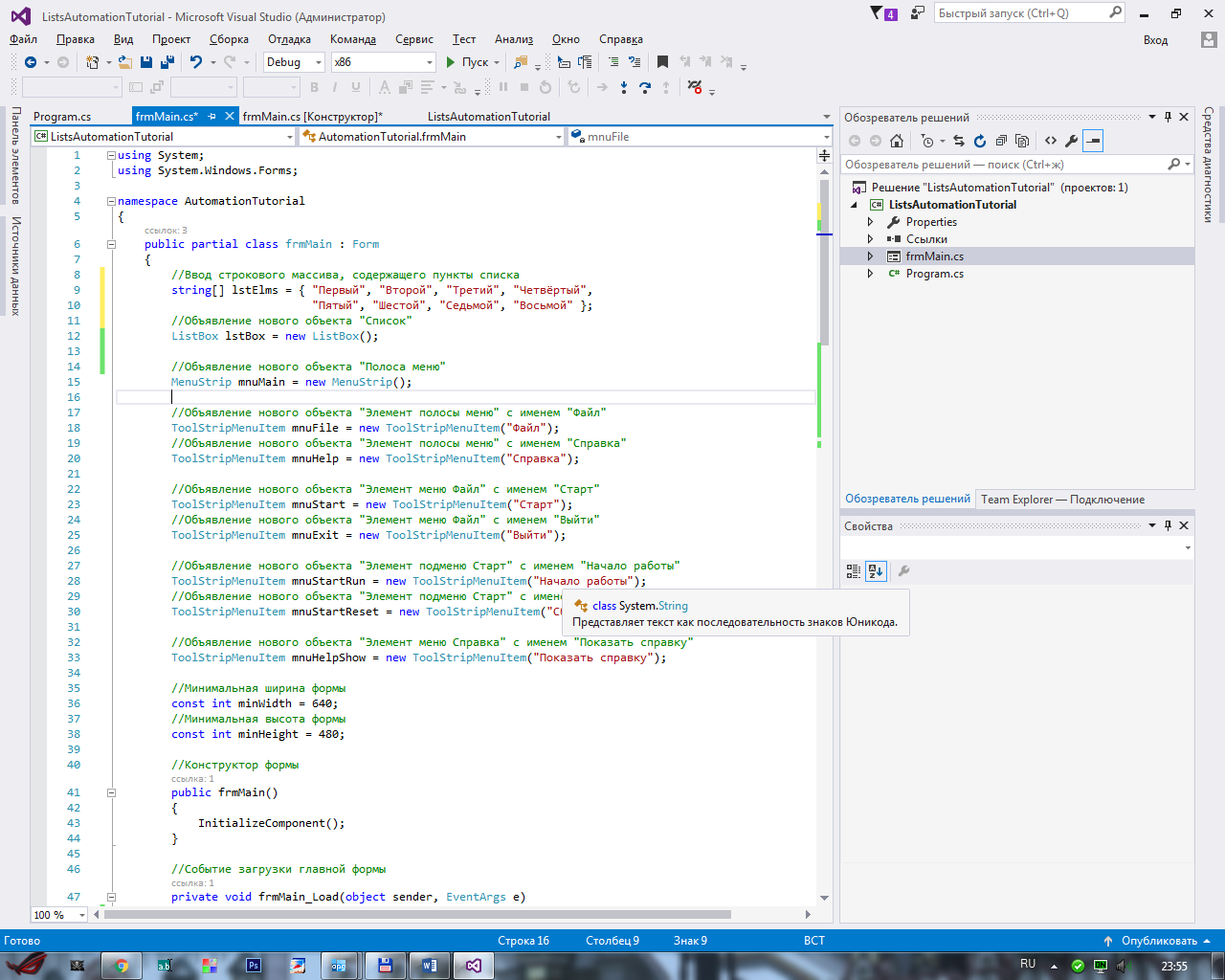


Рисунок 19 – Пример, упрощающий автоматизацию наполнения списка элементами: предварительная запись известных элементов в массив

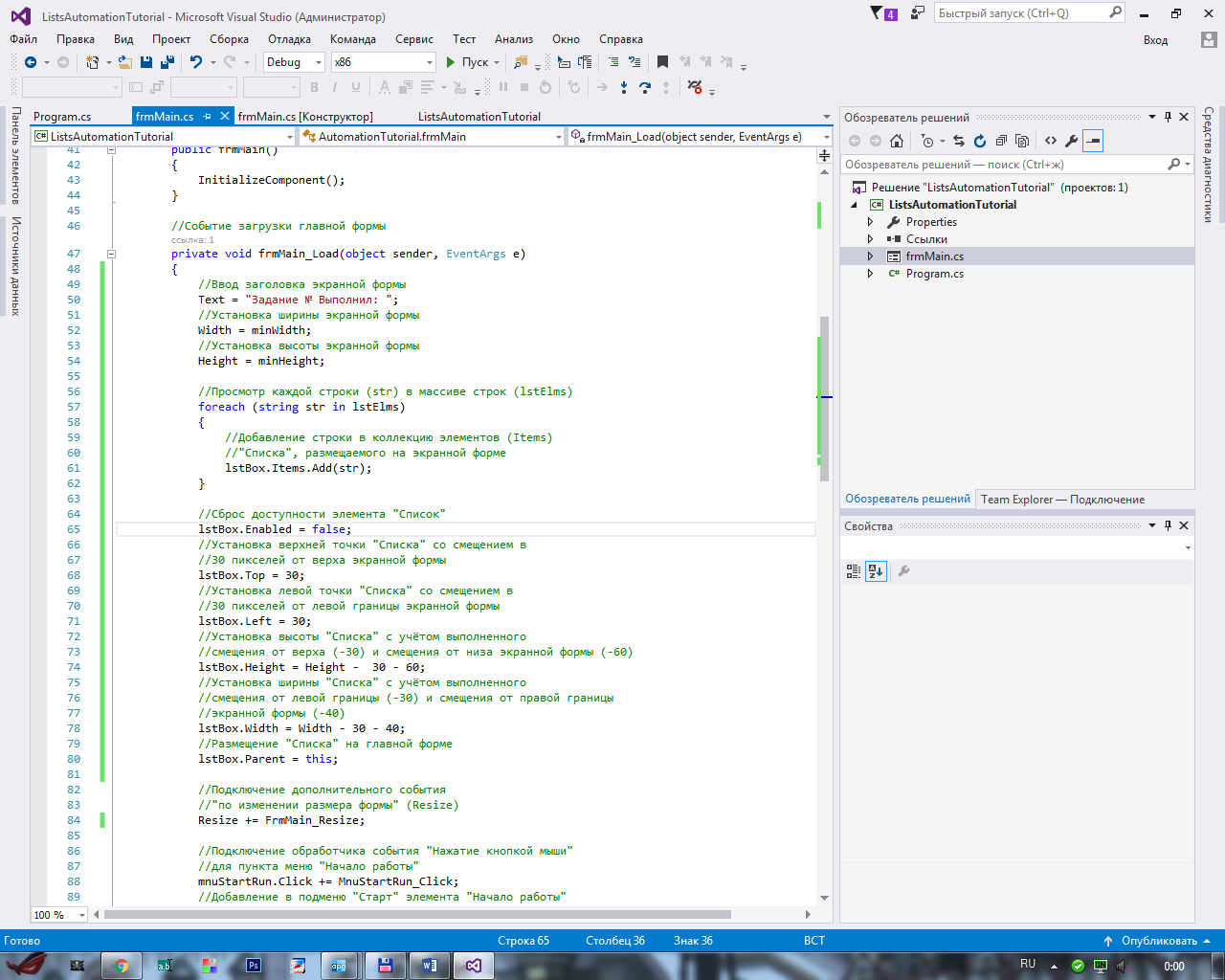


Рисунок 20 – Пример, упрощающий автоматизацию наполнения списка элементами: запуск цикла по каждому элементу массива с добавлением текстового содержимого элемента в список

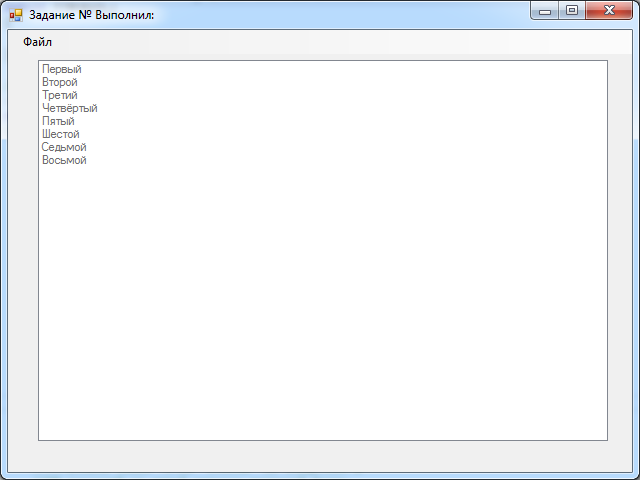


Рисунок 21 – Результат заполнения списка известным элементами в известном количестве

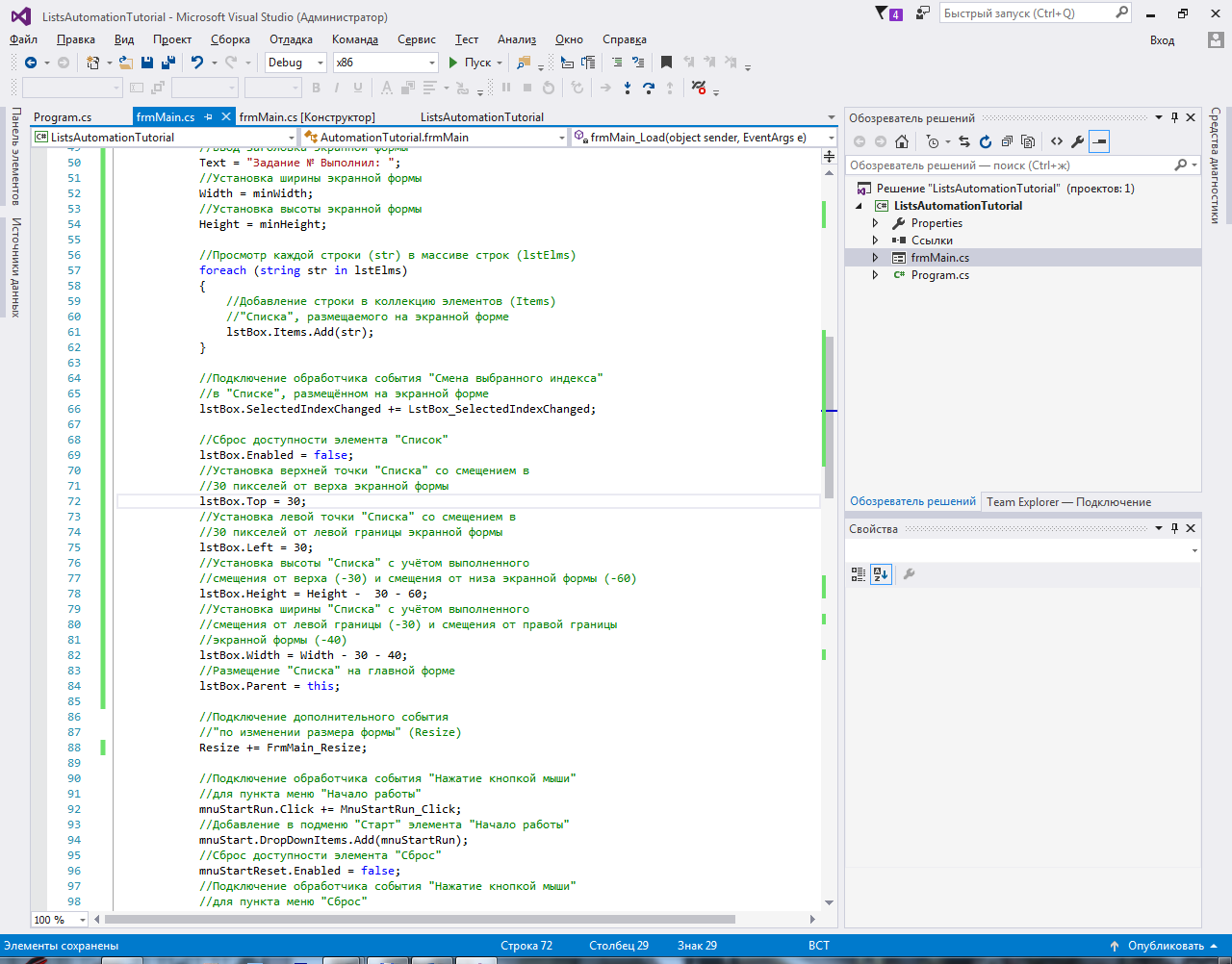


Рисунок 22 – Подключение обработчика события, связанного с изменением индекса в списке

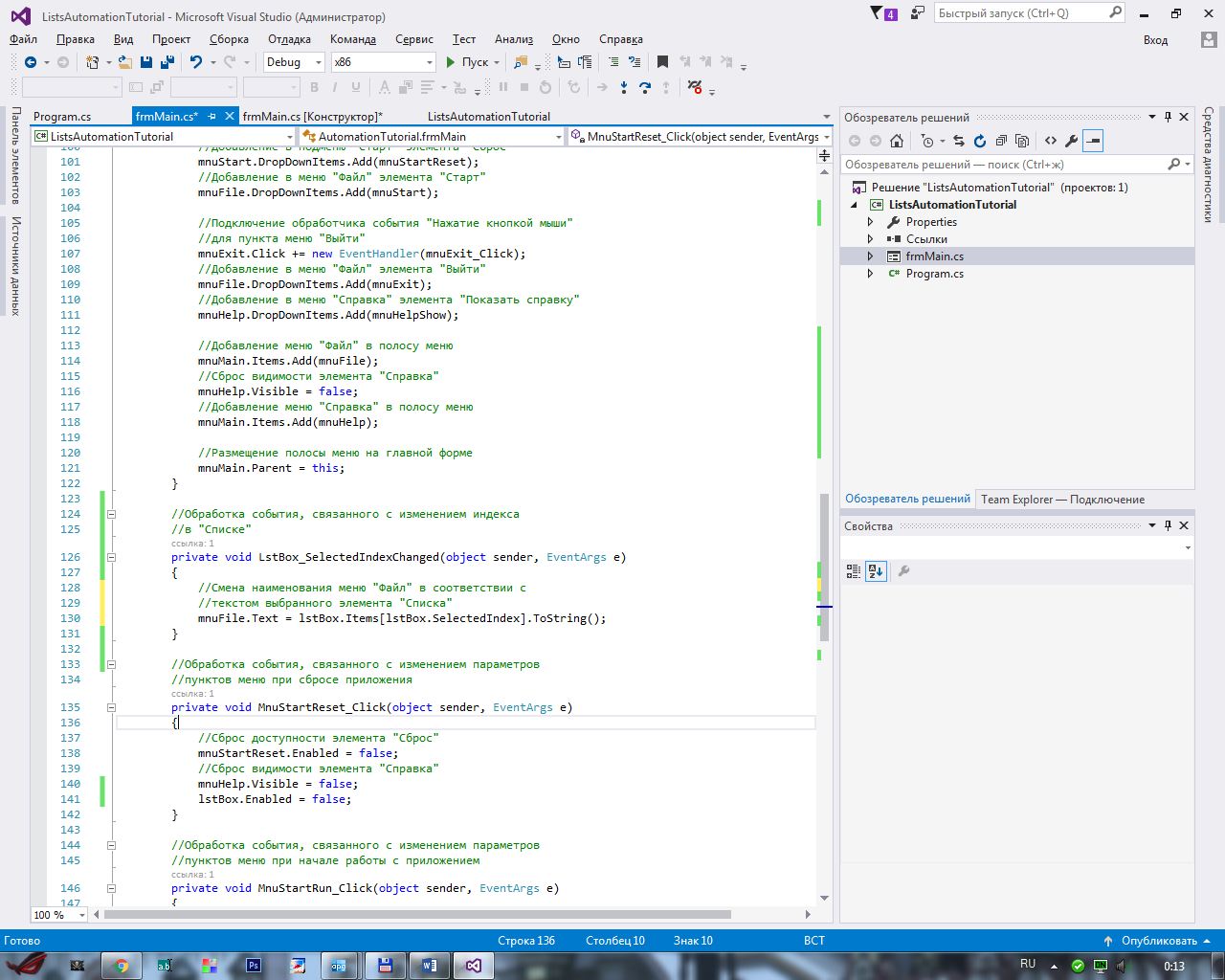


Рисунок 23 – Задача обработчика, например, состоит в замене имени пункта меню выбранным из списка текстовым значением

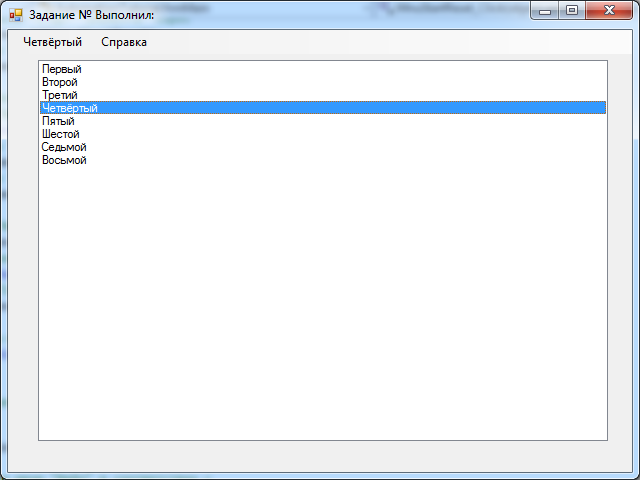


Рисунок 24 – Результат работы приведённой в пример программы