Министерство образования и науки, молодежи и спорта Украины
Донбасская государственная машиностроительная академия (ДГМА)

С. Л. Загребельный,
А. А. Костиков,
В. Э. Миринский

ОСНОВЫ ВИЗУАЛЬНОГО ПРОГРАММИРОВАНИЯ В СРЕДЕ VISUAL STUDIO 2010

Учебное пособие

для студентов специальности 8.05070204
«Электромеханические системы
автоматизации и электропривод»

Краматорск
ДГМА
2012

УДК 004.43
ББК 32.973.26-018.1
З-14

Рецензенты:

Зайцев Д. А., д-р техн. наук, профессор кафедры компьютерной инженерии Международного гуманитарного университета;

Тедеев А. Ф., д-р физ.-мат. наук, зав. отд. «Уравнения математической физики» института Прикладной математики и механики Национальной академии наук Украины.

Розглянуто теоретичні відомості з візуального програмування на мові С++ для підготовки студентів спеціальності 8.05070204 «Електромеханічні системи автоматизації та електропривід». Детально описано теоретичні відомості візуальних компонентів Windows Forms, розглянуто приклади програм і показано хід виконання лабораторних робіт. Кожна лабораторна робота містить контрольні питання для самоперевірки. Наведено завдання для самостійної роботи студентів.

Загребельный, С. Л.

З-14 Основы визуального программирования в среде Visual Studio 2010 : учебное пособие для студентов специальности 8.05070204 «Электромеханические системы автоматизации и электропривод» / С. Л. Загребельный, А. А. Костиков, В. Э. Миринский. – Краматорск : ДГМА, 2012. – 160 с.

ISBN 978-966-379-599-7

Рассмотрены теоретические сведения по визуальному программированию на языке С++ для подготовки студентов специальности 8.05070204 «Электромеханические системы автоматизации и электропривод». Подробно описаны теоретические сведения визуальных компонентов Windows Forms, рассмотрены примеры программ и показан ход выполнения лабораторных работ. Приведены задания для самостоятельной работы студентов.

УДК 004.43
ББК 32.973.26-018.1

ISBN 978-966-379-599-7

СОДЕРЖАНИЕ

1 ЛАБОРАТОРНАЯ РАБОТА 1. СОЗДАНИЕ ПРОСТЕЙШЕЙ ПРОГРАММЫ НА ЯЗЫКЕ VISUAL C++7 7
1.1 Теоретическая часть 7
1.1.1 Элемент управления Form 7
1.1.1.1 Свойства элемента управления Form 8
1.1.1.2 Методы элемента управления Form 13
1.1.1.3 События элемента управления Form 13
1.1.2 Элемент управления TextBox 13
1.1.2.1 Свойства элемента управления TextBox 13
1.1.2.2 Методы элемента управления Textbox 14
1.1.2.3 События элемента управления TextBox 15
1.1.3 Элемент управления Button 17
1.1.3.1 Свойства элемента управления Button 17
1.1.3.2 Методы элемента управления Button 19
1.1.3.3 События элемента управления Button 19
1.1.4 Элемент управления ListView 20
1.1.4.1 Свойства элемента управления ListView 21
1.1.4.2 Методы элемента управления ListView 21
1.1.4.3 События элемента управления ListView 22
1.1.5 Элемент управления RadioButton 23
1.1.5.1 Свойства элемента управления RadioButton 24
1.1.5.2 Методы элемента управления RadioButton 24
1.1.5.3 События элемента управления RadioButton 25
1.1.6 Элемент управления CheckBox 25
1.1.6.1 Свойства элемента управления CheckBox 25
1.1.6.2 Методы элемента управления CheckBox 26
1.1.6.3 События элемента управления CheckBox 27
1.1.7 Элемент управления ListBox 28
1.1.7.1 Свойства элемента управления ListBox 28
1.1.7.2 Методы элемента управления ListBox 29
1.1.7.3 События элемента управления ListBox 29
1.1.8 Элемент управления GroupBox 30
1.1.8.1 Свойства элемента управления GroupBox 30
1.1.8.2 Методы элемента управления GroupBox 31
1.1.8.3 События элемента управления GroupBox 32
1.1.9 Элемент управления PictureBox 32

1.1.9.1 Свойства элемента управления PictureBox 33
1.1.9.2 Методы элемента управления PictureBox 34
1.1.9.3 События элемента управления PictureBox 34
1.1.10 Элемент управления Label 35
1.1.10.1 Свойства элемента управления Label 35
1.1.10.2 Методы элемента управления Label 36
1.1.10.3 События элемента управления Label 36
1.2 Задание к лабораторной работе 36
1.3 Контрольные вопросы 46
2 ЛАБОРАТОРНАЯ РАБОТА 2. УСЛОВНЫЕ ОПЕРАТОРЫ. ВЫЧИСЛЕНИЕ ЗНАЧЕНИЯ ФУНКЦИИ, ЗАДАННОЙ УСЛОВНО 47
2.1 Теоретические сведения 47
2.1.1 Условный оператор if в языке программирования C++ 47
2.1.2 Конструкция if-else 48
2.1.3 Конструкция if-else if-else if-...-else 48
2.1.4 Оператор switch 49
2.1.5 Условный оператор 50
2.1.6 Оператор break (от английского – прерывать) 50
2.1.7 Оператор continue (от английского – продолжать) 51
2.1.8 Оператор goto 51
2.1.9 Математические функции в языке программирования C++ 53
2.2 Пример выполнения работы 54
2.3 Рабочее задание 61
2.4 Контрольные вопросы 63
3 ЛАБОРАТОРНАЯ РАБОТА 3. ЦИКЛИЧЕСКИЙ АЛГОРИТМ. ТАБУЛИРОВАНИЕ ФУНКЦИИ И ПОИСК ЭКСТРЕМУМОВ 64
3.1 Теоретические сведения 64
3.1.1 Оператор while 64
3.1.2 Оператор for 65
3.1.3 Оператор do-while 66
3.2 Пример выполнения работы 68
3.3 Рабочее задание 73
3.4 Контрольные вопросы 75
4 ЛАБОРАТОРНАЯ РАБОТА 4. ПОСТРОЕНИЕ ГРАФИКА ФУНКЦИИ НА ПРОМЕЖУТКЕ С ОПРЕДЕЛЕННЫМ ШАГОМ 76
4.1 Теоретические сведения 76
4.1.1 Элемент управления Chart 76
4.1.1.1 Свойства элемента управления Chart 78
4.1.1.2 Методы элемента управления Chart 79

4.1.1.3 События элемента управления Chart 79
4.2 Пример выполнения работы 80
4.3 Рабочее задание 84
4.4 Контрольные вопросы 86
5 ЛАБОРАТОРНАЯ РАБОТА 5. ПОНЯТИЕ ОДНОМЕРНОГО МАССИВА. СЕЛЕКТИВНАЯ ОБРАБОТКА ЭЛЕМЕНТОВ МАССИВА 87
5.1 Теоретические сведения 87
5.1.1 Понятие одномерного массива 87
5.1.2 Инициализация массива 88
5.1.3 Селективная обработка элементов массива 88
5.2 Пример выполнения работы 89
5.3 Рабочее задание 93
5.4 Контрольные вопросы 95
6 ЛАБОРАТОРНАЯ РАБОТА 6. МНОГОМЕРНЫЙ МАССИВ. ПОНЯТИЕ МАТРИЦЫ. СЕЛЕКТИВНАЯ ОБРАБОТКА ЭЛЕМЕНТОВ СТРОК, СТОЛБЦОВ И ДИАГОНАЛЕЙ МАТРИЦЫ 96
6.1 Теоретические сведения 96
6.1.1 Понятие матрицы 96
6.1.2 Инициализация матрицы 97
6.1.3 Селективная обработка элементов матрицы 97
6.1.4 Использование элемента управления DataGridView для работы с матрицами 98
6.1.4.1 Использование элемента управления DataGridView в несвязанном режиме 100
6.1.4.2 Персональная настройка элемента управления DataGridView 101
6.1.4.3 Настройка ячеек заголовков 103
6.1.4.4 Настройка ячеек, не являющихся заголовками 103
6.1.4.5 Настройка элемента управления 104
6.1.4.6 Настройка заголовков столбцов 106
6.1.4.7 Форматирование столбца 107
6.1.4.8 Настройка внешнего вида чередующихся строк 108
6.1.4.9 Динамическое определение стилей ячеек 108
6.2 Пример выполнения работы 109
6.3 Рабочее задание 113
6.4 Контрольные вопросы 116
7 ЛАБОРАТОРНАЯ РАБОТА 7. ИЗУЧЕНИЕ ВЕРОЯТНОСТНЫХ АЛГОРИТМОВ 117
7.1 Теоретические сведения 117
7.2 Пример выполнения работы 118

7.3 Рабочее задание 122
8 ЛАБОРАТОРНАЯ РАБОТА 8. РАБОТА С ДИАЛОГОВЫМИ ОКНАМИ. СОЗДАНИЕ ОПЕРАЦИОННОГО МЕНЮ. 125
8.1 Теоретические сведения 125
8.1.1 Понятие операционное меню 125
8.1.2 Обработчик событий для элементов операционного меню) 129
8.1.3 Элемент управления с вкладками (TabControl) 129
8.1.4 Создание диалогового окна 131
8.1.5 Создание обработчика событий диалогового окна 132
8.1.6 Помещение на форму диалоговых окон `ColorDialog`, `FontDialog`, `SaveFileDialog` 133
8.2 Пример выполнения работы 135
8.3 Рабочее задание 146
8.4 Контрольные вопросы 148
9 САМОСТОЯТЕЛЬНАЯ РАБОТА. ОБРАБОТКА МАТРИЦЫ. ФОРМИРОВАНИЕ ОДНОМЕРНЫХ МАССИВОВ ИЗ ДВУМЕРНЫХ. СОРТИРОВКА ОДНОМЕРНЫХ МАССИВОВ. 149
9.1 Теоретические сведения 149
9.1.1 Особенности работы с матрицами 149
9.1.2 Сортировка одномерных массивов 150
9.1.2.1 Сортировка методом «пузырька» 150
9.1.2.2 Сортировка методом перестановки 151
9.1.2.3 Сортировка методом вставки 151
9.1.2.4 Сортировка методом Шелла 153
9.2 Рабочее задание 153
Контрольные вопросы 155

СПИСОК ЛИТЕРАТУРЫ 156

Приложение А. График сдачи модулей для студентов дневной формы обучения 157

Приложение Б. Состав модулей дисциплины «Вычислительная техника и программирование»для студентов специальности 8.05070204 «Электромеханические системы автоматизации и электропривод» дневной формы обучения, распределение времени на их усвоение, термины контроля 158

1 ЛАБОРАТОРНАЯ РАБОТА 1. СОЗДАНИЕ ПРОСТЕЙШЕЙ ПРОГРАММЫ НА ЯЗЫКЕ VISUAL C++

Цель: научиться использовать среду Visual Studio 2010 для создания простейших windows приложений по программированию на языке C++.

1.1 Теоретическая часть

1.1.1 Элемент управления `Form`

Основной частью визуального приложения является форма. Поэтому при программировании очень важно уделять внимание внешнему виду форм и функциям. Итак, форма представляет собой пустую доску, которую разработчик оснащает элементами управления, формируя интерфейс пользователя, и кодом для управления данными. Для этого `Visual Studio` обеспечивает интегрированную среду разработки, способствующую написанию кодов, а также расширенный набор элементов управления `.NET Framework`. Дополняя функциональными возможностями этих элементов управления свои коды, пользователь может легко и быстро разработать необходимое приложение.

При создании проекта в `Visual Studio` создается объект, называемый формой. Пример формы показан на рис. 1.1.

Рисунок 1.1 – Пример окна

Форма – это главный контейнер, на котором размещаются компоненты среды разработки (см. рис. 1.2).

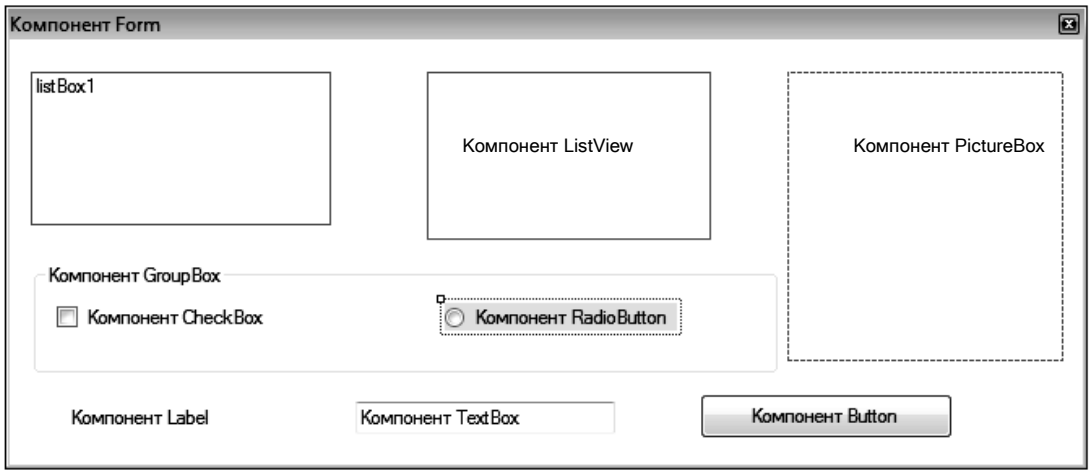


Рисунок 1.2 – Форма с размещенными компонентами

Как и любой объект, форма имеет свойства, методы и события. Для вызова свойств любого элемента его сначала нужно выделить, а затем нажать на кнопку  на панели инструментов, вследствие чего появится следующее окно (см. рис. 1.3).

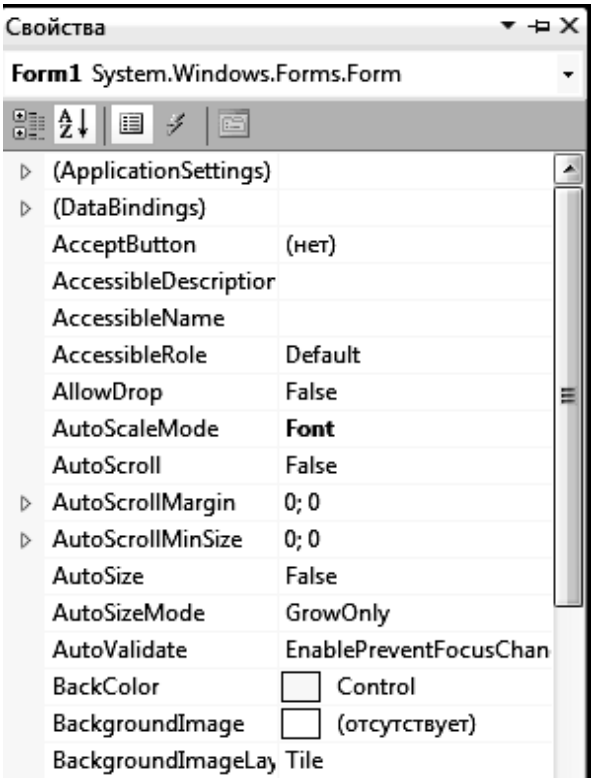


Рисунок 1.3 – Окно «Свойства»

1.1.1.1 Свойства элемента управления `Form`. Основные свойства формы отображены в таблице 1.1

Таблица 1.1 – Основные свойства Form

Название свойства	**Описание свойства**
1	2
BackgroundColor	Цвет фона окна. Данное свойство имеет тип класса `color`. Если щелкнуть в поле напротив свойства, то появится кнопка выпадающего списка. Щелкните по ней, и вы увидите панель выбора цвета. Панель состоит из трех закладок: `Custom` (*Произвольные*), `Web` (*Web-цвета*) и `System` (*Системные*). Для окон желательно указывать один из системных цветов и лучше, если это будет `control` для диалоговых окон и `window` для окон, которые должны отображать данные
Cursor	Позволяет задать курсор, который будет отображаться в момент, когда курсор мыши находится над поверхностью окна. При изменении этого свойства в визуальном дизайнере появляется выпадающий список, в котором перечислены названия типов курсоров и даже показаны их изображения
Enabled	Позволяет задать доступность окна. Если свойство равно `false`, то компонент не будет реагировать на действия пользователя, т. е. будет недоступен
Font	Шрифт, который будет использоваться для вывода текста поверх окна. Для окна это на первый взгляд не очень актуально. Но большая актуальность этого свойства проявляется у компонентов, которые содержат надписи. Чтобы изменить шрифт, можно поступить двумя способами: − выделить строку со свойством `Font`, и в правом углу появится кнопка с тремя точками. Щелкните по ней, и вы увидите стандартное окно задания свойств шрифта; − свойство шрифта на самом деле является самостоятельным классом `Font` со своими свойствами. Вы можете редактировать их каждое в отдельности. Для этого нужно щелкнуть по кнопке с крестиком слева от имени свойства. В окне `Properties` свойства формы расступятся, уступив место свойствам шрифта
ForeColor	Цвет переднего плана, который чаще всего используется в качестве цвета текста. Если вы посмотрите на доступные свойства шрифта, то заметите, что среди них нет цвета. Цвет текста задается через `ForeColor`

Продолжение таблицы 1.1

1	2
FormBorderStyle	Стиль контура формы. Это свойство является перечислением, в редакторе свойств появляется выпадающий список для выбора из следующих элементов (рекомендуем на реальном приложении попробовать установить каждое из свойств и посмотреть результат лично): – `None` – контура не будет; – `FixedSingle` – тоненький фиксированный контур, не позволяющий изменять размеры окна; – `Fixed3D` – контур фиксированный, не позволяющий изменять размеры окна, но трехмерный, создающий эффект утопления окна; – `FixedDialog` – похож на `Fixedsingle`, только в заголовке окна не будет иконки. Такой контур чаще всего используют для диалоговых окон; – `Sizable` – стандартный контур, позволяющий изменять размеры окна; – `FixedToolWindow` – контур с очень тоненьким заголовком окна, не позволяющий изменять размеры окна. Такие окна чаще всего используются для окон с кнопками, например, окно инструментов в Abode Photoshop; – `SizableToolWindow` – то же самое, что и `FixedToolWindow`, только с возможностью изменять размер окна
Icon	В этом свойстве можно задать иконку для формы. Здесь тоже есть кнопка, которая вызывает стандартное окно для открытия файла. Загруженная иконка попадет в resx-файл формы
Location	Положение окна относительно левого верхнего угла экрана. Такое же свойство есть и у компонентов, но там расстояние отсчитывается от левого верхнего угла компонента родителя, на поверхности которого находится наш элемент управления. Данное свойство состоит из двух составляющих – X и Y, которые раскрываются, если щелкнуть по кнопке с крестиком слева от имени свойства
MainMenustrip	В этом свойстве можно задать компонент меню, который будет использоваться в качестве главного меню для окна. Для этого на форме должен стоять компонент класса `Menustrip`. Тогда этот компонент можно будет выбрать из выпадающего списка

Продолжение таблицы 1.1

1	2
MaximizeBox	Булево значение, которое определяет, должна ли отображаться в заголовке окна кнопка максимизации окна. Если у вас выбран стиль окна, не позволяющий изменять размеры окна, то убедитесь, что это свойство равно false, чтобы спрятать кнопку. Несмотря на то, что форма не разрешает изменение размеров, кнопка максимизации без проблем распахнет окошко на полный экран
MaximumSize	Позволяет задать максимальные ширину и высоту окна. Пользователь не сможет растянуть окно больше, чем указанные размеры
MinimizeBox	Булево значение, определяющее, нужно ли отображать кнопку сворачивания окна
MinimumSize	Минимальные размеры окна, менее которых окно не может быть уменьшено. Такое ограничение может пригодиться, если слишком маленькие размеры портят дизайн окна
Name	Название формы и имя переменной объекта, которую вы можете использовать в коде для доступа к свойствам объекта и для вызова его методов
Opacity	Процент непрозрачности окна. По умолчанию свойство равно 100 %, т. е. окно абсолютно не прозрачно. Если нужно наполовину просветить задний план, то можно установить значение в 50 %
Padding	Отступы от краев окна. Свойство состоит из пяти значений: `All`, `Left`, `Top`, `Right`, `Bottom`, что соответствует отступам со всех сторон: слева, сверху, справа и снизу. Если установить левый отступ в значение 10, то левая кромка окна будет размером в 10 пикселов
Showicon	Булево значение, определяющее, нужно ли показывать иконку в заголовке окна
ShowinTaskBar	Нужно ли отображать окно в Панели задач. Главное окно чаще всего нужно отображать в Панели задач, а вот для дочерних и диалоговых окон это свойство лучше отключать, иначе для одного приложения для разных окон в Панели задач будет создаваться несколько меток. Как правило, этого не должно быть
Size	Размер окна, его ширина и высота

Продолжение таблицы 1.1

1	2
StartPosition	Начальная позиция окна. У этого свойства есть выпадающий список для выбора одного из следующих значений: – `WindowsDefaultLocation` – положение окна определяется ОС; – `WindowsDefauitBounds` – система будет определять не только положение, но и размеры окна; – `CenterParent` – расположить окно по центру родительского окна. Это значение лучше всего использовать для всех дочерних окон; – `Manual` – расположение будет задаваться программистом самостоятельно через свойство `Location`; – `CenterScreen` – отцентрировать по центру экрана
Tag	Очень интересное свойство, но оно абсолютно ни на что не влияет. Но зачем тогда оно нужно? Вы можете сохранять в этом свойстве любую дополнительную информацию, потому что это свойство имеет тип `object`
Text	Текстовый заголовок окна
TopMost	Булево значение, которое определяет, должно ли окно располагаться поверх других окон. Не злоупотребляйте этим свойством в своих программах для главного окна, а если и решите расположить окно поверх других, то дайте пользователю возможность отключать эту особенность. Многих пользователей раздражают окна, которые перекрывают рабочую область, а вот использование этого свойства для дочерних окон приложения может быть очень удобным. Если окно показано не модально, это свойство позволит окну быть поверх остальных и не спрятаться за другими окнами
Windowstate	Состояние окна. Здесь можно указать одно из трех значений: – `Normal` – нормальное состояние; – `Maximized` – окно должно быть развернуто на весь экран; – `Minimized` – окно должно быть минимизировано.

1.1.1.2 Методы элемента управления `Form`

Основные методы формы (`Form`) отображены в таблице 1.2.

Таблица 1.2 – Основные методы `Form`

Название метода	Описание метода
`Close()`	Закрывает форму
`Hide()`	Форма становится невидимой
`Show()`	Выводит форму на экран
`ShowDialog()`	Показывает форму в модальном режиме
`Dispose()`	Форма разрушается и память, занятая ею, освобождается
`Focus()`	Делает форму активной

1.1.1.3 События элемента управления `Form`

Основные события формы отображены в таблице 1.3.

Таблица 1.3 – Основные события `Form`

Название события	Описание события
`Activated`	Возникает, когда форма активизирована
`Click`	Возникает при щелчке мышью на форме
`Load`	Возникает перед первым выводом формы

1.1.2 Элемент управления `TextBox`

Компонент находится в списке `All Windows Forms` палитры компонентов. Он задает многострочное или однострочное редактируемое поле, через которые вводят (выводят) строчные данные.

1.1.2.1 Свойства элемента управления `TextBox`

Основные свойства элемента управления `TextBox` отображены в таблице 1.4

Таблица 1.4 – Основные свойства TextBox

Название свойства	Описание свойства
AcceptsReturn	Получает или задает значение, указывающее, что происходит в многострочном элементе управления TextBox при нажатии клавиши ENTER: создается новая строка текста или активируется кнопка стандартного действия формы
AcceptsTab	Получает или задает значение, указывающее, что происходит при нажатии клавиши TAB в многострочном элементе управления: вводится знак табуляции в текстовом поле или фокус ввода в форме перемещается к следующему элементу управления в последовательности переходов
Lines	Получает или задает строки текста в элементе управления «Текстовое поле»
Multiline	Получает или задает значение, показывающее, является ли данный элемент управления многострочным TextBox
PasswordChar	Получает или задает знак, используемый для маскировки знаков пароля, вводимых в однострочный элемент управления TextBox
ReadOnly	Получает или задает значение, указывающее, является ли текст в текстовом поле доступным только для чтения
ScrollBars	Получает или задает значение, показывающее, какие полосы прокрутки должны присутствовать в многострочном элементе управления TextBox
Text	Получает или задает текущий текст в текстовом поле TextBox
TextAlign	Получает или задает способ выравнивания текста в элементе управления TextBox

1.1.2.2 Методы элемента управления Textbox

Основные методы объекта Textbox отображены в таблице 1.5

Текст, отображаемый в элементе управления, содержится в свойстве Text. По умолчанию в текстовом поле можно ввести до 2048 знаков. Если свойству Multiline присвоить значение true, это позволит вводить до 32 килобайт текста. Свойство Text может быть установлено в окне «Свой-

ства» во время разработки, программными средствами во время выполнения или в результате ввода данных пользователем во время выполнения.

Текущее содержимое текстового поля может быть получено во время выполнения путем считывания значения свойства `Text`.

В следующем примере кода текст помещается в элемент управления во время выполнения. Процедура `InitializeMyControl` не выполняется автоматически, она должна быть вызвана.

```
private:
void InitializeMyControl()
{
// Запись текста в элемент управления TextBox, имеющего имя TextBox1.
textBox1->Text = "Это элемент управления TextBox.";
}
```

Таблица 1.5 – Основные методы `TextBox`

Название метода	Описание метода
`AppendText`	Добавляет строку к содержимому текстового элемента управления
`Clear`	Удаляет из текстового поля все его содержимое
`Copy`	Копирует текущее выделение текста в элементе управления, поддерживающем редактирование текста
`CreateGraphics`	Задает объект `Graphics` для элемента управления
`Cut`	Перемещает текущий выбор из текстового поля в буфер обмена
`DeselectAll`	Указывает, что значение свойства `SelectionLength` равно нулю для отмены выделения символов в элементе управления
`Dispose()`	Освобождает все ресурсы, используемые объектом
`Focus`	Задает фокус ввода элемента управления
`Hide`	Скрывает элемент управления от пользователя
`Paste()`	Заменяет текущий выбор в текстовом поле содержимым буфера обмена
`Select()`	Активирует элемент управления
`SelectAll`	Выбирает весь текст в текстовом поле
`Show`	Отображает элемент управления для пользователя
`Undo`	Отменяет последнюю операцию редактирования в текстовом поле

1.1.2.3 События элемента управления `TextBox` .Основные события элемента управления `TextBox` отображены в таблице 1.6.

Таблица 1.6 – Основные события `TextBox`

Название события	Описание события
GotFocus	Событие, возникающее в момент активизации окна
LostFocus	Событие, возникающее в момент потери фокуса
KeyDown	Событие, возникающее в момент движения нажимаемой клавиши вниз
KeyPress	Событие, возникающее при удержании нажатой клавиши
KeyUp	Событие, возникающее при отпускании нажатой клавиши
Change	Событие, возникающее при изменении, добавлении или удалении очередного символа в поле ввода

Рассмотрим пример использования события `KeyDown` для определения типа символа, введенного в элемент управления.

```
// Булево значение flag используется для определения символа, отличного от числа
     private:
     bool nonNumberEntered;
// Обработка события KeyDown для определения символа, введенного в элемент управления
     void textBox1_KeyDown (Object^, System: :Windows: :Forms:
     :KeyEventArgs^ e )
   {
// Присваивает значение переменной flag равным false
nonNumberEntered = false;
// Определяет, является ли введенный символ числом, введенным с верхней части клавиатуры
     if ( e->KeyCode < Keys::D0 || e->KeyCode > Keys::D9 )
      {
// Определяет, является ли введенный символ числом, введенным с цифровой части клавиатуры
     if (e->KeyCode<Keys::NumPad0||e->KeyCode> Keys::NumPad9)
         {
// Определяет, является ли введенный символ символом Backspace
     if ( e->KeyCode != Keys::Back )
      {
// Введен нецифровой символ
// Устанавливается flag в true и анализируется в событии KeyPress
     nonNumberEntered = true;
            }
         }
      }
// Если клавиша Shift нажата, это не число
     if (Control::ModifierKeys == Keys::Shift) {
     nonNumberEntered = true;
      }
   }
// Это событие происходит после события KeyDown и может быть использовано для
предотвращения ввода символов в элемент управления
```

```
    void     textBox1_KeyPress(Object^,System::Windows::Forms:
    :KeyPressEventArgs^ e )
  {
// Проверяет flag, установленный в событии KeyDown
    if ( nonNumberEntered == true )
    {
// Прекращает ввод нечислового символа в элемент управления
        e->Handled = true;
     }
  }
```

1.1.3 Элемент управления **`Button`**

Элемент управления `Windows Forms Button` (кнопка) служит для выполнения действия с помощью мыши. На элементе управления `Button` может отображаться как текст, так и рисунок. Если нажать кнопку, она выглядит так, словно она нажата и отпущена. При нажатии на кнопку вызывается обработчик событий `Click`. В обработчик событий `Click` помещается код, отвечающий за выполнение нужного действия.

Текст, отображающийся на кнопке, содержится в свойстве `Text`. Если текст превышает ширину кнопки, он переходит на следующую строку. Однако если высота элемента управления не может изменяться, текст обрезается. Свойство `Text` может содержать клавишу доступа, что позволяет пользователю выполнять действие, аналогичное щелчку элемента управления, нажав клавишу `ALT` одновременно с клавишей доступа. Внешний вид текста управляется свойством `Font` и свойством `TextAlign`.

В элементе управления `Button` можно также отображать рисунки с помощью свойств `Image` и `ImageList`.

1.1.3.1 Свойства элемента управления `Button`

Основные свойства элемента управления `Button` отображены в таблице 1.7

Таблица 1.7 – Основные свойства `Button`

Название свойства	**Описание свойства**
1	2
`Anchor`	Возвращает или задает границы контейнера, с которым связан элемент управления, и определяет способ изменения размеров элемента управления при изменении размеров его родительского элемента

Продолжение таблицы 1.7

1	2
AutoEllipsis	Получает или задает значение, указывающее, отображается ли знак многоточия (...) в правом углу элемента управления, обозначающий, что текст элемента управления выходит за пределы указанной длины этого элемента
AutoSize	Получает или задает значение, указывающее, основано ли изменение размеров элемента управления на его содержимом
Capture	Возвращает или задает значение, определяющее, была ли мышь захвачена элементом управления
DialogResult	Возвращает или задает значение, возвращаемое в родительскую форму при нажатии кнопки
Dock	Возвращает или задает границы элемента управления, прикрепленные к его родительскому элементу управления, и определяет способ изменения размеров элемента управления с его родительским элементом управления
FlatAppearance	Возвращает внешний вид границ и цвета, используемые для определения состояния флажка и состояние мыши
FlatStyle	Получает или задает плоский внешний вид для кнопки
Image	Получает или задает изображение, отображаемое в кнопке
ImageAlign	Получает или задает выравнивание изображения в кнопке
ImageIndex	Получает или задает значение индекса списка изображений для изображения, отображаемого в кнопке
TabIndex	Возвращает или задает последовательность перехода элемента управления внутри контейнера
TabStop	Получает или задает значение, показывающее, может ли пользователь перевести фокус в данный элемент управления при помощи клавиши TAB
TextImageRelation	Получает или задает метаположение текста и изображения относительно друг друга

Продолжение таблицы 1.7

UseMnemonic	Получает или задает значение, которое указывает, должен ли первый знак, следующий за знаком амперсанда (&), использоваться как мнемонический ключ элемента
UseVisualStyleBackColor	Получает или задает значение, которое указывает, должен ли фон рисоваться с использованием стилей оформления (если они поддерживаются)
Visible	Получает или задает значение, указывающее, отображаются ли элемент управления и все его дочерние элементы управления

1.1.3.2 Методы элемента управления `Button`

Основные методы элемента управления `Button` отображены в таблице 1.8

Таблица 1.8 – Основные методы `Button`

Название метода	Описание метода
Hide	Скрывает элемент управления от пользователя
Focus	Задает фокус ввода элемента управления
Select()	Активирует элемент управления
Show	Отображает элемент управления для пользователя

1.1.3.3 События элемента управления `Button`

Основные события элемента управления `Button` отображены в таблице 1.9

Таблица 1.9 – Основные события `Button`

Название события	Описание события
Click	Происходит при щелчке элемента управления
Enter	Происходит при входе в элемент управления
MouseHover	Происходит, когда указатель мыши задерживается на элементе управления
MouseLeave	Происходит, когда указатель мыши покидает элемент управления

Чаще всего элемент управления Button в Windows Forms используется для выполнения какой-либо программы при нажатии кнопки.

Щелчок элемента управления Button вызывает также некоторые другие события, например MouseEnter, MouseDown и MouseUp. Если требуется вложить обработчики событий для таких событий, связанных с основным, нужно убедиться, что их действия не конфликтуют. Например, если нажатие кнопки удаляет сведения, введенные пользователем в текстовое поле, при наведении указателя мыши на кнопку не должна появляться подсказка с несуществующими сведениями.

Если дважды щелкнуть элемент управления Button, каждый щелчок будет обрабатываться отдельно; другими словами, этот элемент управления не поддерживает событие двойного щелчка.

Пример. Для того, чтобы при нажатии на кнопку появилось сообщение, необходимо написать следующий обработчик события Click кнопки с именем Button1.

```
private:
   void button1_Click(System::Object ^ sender,
      System::EventArgs ^ e)
   {
      MessageBox::Show("button1 нажата");
   }
```

1.1.4 Элемент управления `ListView`

В элементе управления Windows Forms ListView отображается список элементов со значками. Представление в виде списка может использоваться для создания пользовательского интерфейса, аналогичного правой области окна Windows Explorer.

Для этого элемента управления предусмотрено четыре режима представления: LargeIcon (крупные значки), SmallIcon (мелкие значки), List (список) и Details (таблица). В режиме LargeIcon рядом с текстом элементов отображаются крупные значки; если элемент управления достаточно большой, элементы появляются в нескольких столбцах. Режим SmallIcon аналогичен предыдущему, за исключением того, что отображаются мелкие значки. В режиме List отображаются мелкие значки, но всегда в одном столбце. В режиме Details элементы отображаются в нескольких столбцах.

1.1.4.1 Свойства элемента управления `ListView`

Основные свойства элемента управления `ListView` отображены в таблице 1.10

Таблица 1.10 – Основные свойства `ListView`

Название свойства	Описание свойства
`Items`	Получает коллекцию, содержащую все элементы в данном элементе управления
`SelectedItems`	Получает элементы, выбранные в данном элементе управления
`MultiSelect`	Возвращает или задает значение, указывающее, возможен ли выбор нескольких элементов
`CheckBoxes`	Возвращает или задает значение, указывающее, будет ли в данном элементе управления отображаться флажок рядом с каждым элементом
`StateImageList`	Возвращает или задает список `ImageList`, сопоставленный с устанавливаемыми приложением состояниями в этом элементе управления
`Activation`	Возвращает или задает тип действия, который должен выполнить пользователь для активации элемента
`HotTracking`	Возвращает или задает значение, указывающее, принимает ли текст элемента или подэлемента форму гиперссылки, когда на него наводится указатель мыши
`GridLines`	Возвращает или задает значение, показывающее, будут ли видны в элементе управления линии сетки между строками и столбцами, содержащими элементы и подэлементы
`LabelEdit`	Возвращает или задает значение, указывающее, может ли пользователь изменять метки элементов в данном элементе управления (надо щелкнуть текст элемента, чтобы перевести его в состояние редактирования)
`Groups`	Получает коллекцию объектов `ListViewGroup`, назначенную элементу управления

1.1.4.2 Методы элемента управления `ListView`

Основные методы элемента управления `ListView` отображены в таблице 1.11

Таблица 1.11 – Основные методы `ListView`

Название метода	Описание метода
`Clear`	Удаляет все элементы и столбцы из данного элемента управления
`Contains`	Получает значение, показывающее, является ли указанный элемент управления дочерним элементом
`CreateGraphics`	Задает объект `Graphics` для элемента управления
`BeginUpdate`	Запрещает прорисовку элемента управления до вызова метода `EndUpdate`
`EndUpdate`	Возобновляет прорисовку элемента управления списка после того, как она была приостановлена методом `BeginUpdate`
`Hide`	Скрывает элемент управления от пользователя
`Select()`	Активирует элемент управления
`SelectNextControl`	Активирует следующий элемент управления
`Show`	Отображает элемент управления для пользователя
`Sort`	Сортирует элементы в представлении списка
`Update`	Вызывает перерисовку элементом управления недопустимых областей клиентской области

1.1.4.3 События элемента управления `ListView`

Основные события элемента управления `ListView` отображены в таблице 1.12.

Таблица 1.12 – Основные события `ListView`

Название события	Описание события
`Click`	Происходит при щелчке элемента управления
`ItemActivate`	Происходит при активизации элемента
`ItemCheck`	Происходит при изменении состояния флажка элемента
`ItemMouseHover`	Происходит при наведении указателя мыши на элемент
`ItemSelectionChanged`	Происходит при изменении состояния выбора элемента
`Validated`	Происходит по завершении проверки элемента управления
`Validating`	Происходит при проверке элемента управления

Основным свойством элемента управления ListView является свойство Items, содержащее элементы, отображаемые элементом управления. Свойство SelectedItems содержит коллекцию элементов, выбранных в настоящий момент в элементе управления. Если для свойства MultiSelect задано значение true, пользователь имеет возможность выбрать несколько элементов, например, перетащить сразу несколько элементов в другой элемент управления. Если для свойства CheckBoxes задано значение true, в элементе управления ListView рядом с элементами могут отображаться флажки.

Свойство Activation определяет тип действия, которое должен выполнить пользователь, чтобы активировать элемент в списке. Возможные варианты: Standard, OneClick и TwoClick. Активация OneClick означает, что для активации элемента требуется сделать один щелчок. Для активации TwoClick требуется, чтобы пользователь дважды щелкнул активируемый элемент; одиночный щелчок изменит цвет текста элемента. Для активации Standard требуется, чтобы пользователь дважды щелкнул активируемый элемент, но внешний вид элемента при этом не изменяется.

Элемент управления ListView также поддерживает визуальные стили и другие возможности, предоставляемые платформой Windows XP, включая группирование, мозаичное представление и метки вставки.

1.1.5 Элемент управления RadioButton

Элемент управления Windows Forms RadioButton (переключатель) обеспечивает выбор из двух или более взаимоисключающих вариантов. Функции переключателей и флажков могут показаться схожими, но между ними есть важное отличие: в случае переключателя пользователь может выбрать лишь один вариант. Однако флажков можно выбрать любое количество. Определяя группу значений переключателя, разработчик формы предлагает пользователю набор вариантов, из которых может быть задан один и только один.

При щелчке элемента управления RadioButton, его свойству Checked присваивается значение true и вызывается обработчик событий Click. При изменении значения свойства Checked происходит событие CheckedChanged. Если свойство AutoCheck имеет значение true (принимается по умолчанию), то при выборе одного значения переключателя остальные значения группы автоматически сбрасываются. Обычно этому свойству присваивают значение false только в тех случаях, когда в коде предусмотрена проверка допустимости выбранного варианта переключателя. Текст, связанный с этим элементом управления, задается свойством Text, которое также может определять клавиши быстрого доступа. Клавиша доступа позволяет пользователю щелкнуть другой элемент управления,

используя сочетание клавиши ALT и заданной клавиши. Элемент управления RadioButton может выглядеть как кнопка команды, которая отображается как нажатая при выбранном значении переключателя, если свойство Appearance имеет значение Button. В переключателях можно также отображать рисунки с помощью свойств Image и ImageList.

1.1.5.1 Свойства элемента управления RadioButton

Основные свойства элемента управления RadioButton отображены в таблице 1.13

Таблица 1.13 – Основные свойства RadioButton

Название свойства	Описание свойства
Appearance	Получает или задает значение, определяющее внешний вид переключателя RadioButton
AutoCheck	Получает или задает значение, показывающее, будет ли автоматически изменяться значение Checked и внешний вид элемента управления, когда он выбирается щелчком
Image	Получает или задает изображение, отображаемое в кнопке
ImageList	Получает или задает свойство ImageList, содержащее изображение Image, отображенное в кнопке
Checked	Получает или задает значение, показывающее, выбран ли данный элемент управления

1.1.5.2 Методы элемента управления RadioButton

Основные методы элемента управления RadioButton отображены в таблице 1.14.

Таблица 1.14 – Основные методы RadioButton

Название метода	Описание метода
Contains	Получает значение, показывающее, является ли указанный элемент управления дочерним элементом
CreateGraphics	Задает объект Graphics для элемента управления
Focus	Задает фокус ввода элемента управления
IsInputChar	Определяет, является ли символ входным символом, который распознается элементом управления
IsInputKey	Определяет, является ли заданная клавиша обычной клавишей ввода или специальной клавишей, нуждающейся в предварительной обработке

1.1.5.3 События элемента управления `RadioButton`

Основные события элемента управления `RadioButton` отображены в таблице 1.15.

Таблица 1.15 – Основные события `RadioButton`

Название события	Описание события
`CheckedChanged`	Происходит при изменении значения свойства `Checked`
`Click`	Происходит при щелчке элемента управления
`Disposed`	Происходит при удалении компонента вызовом метода `Dispose`
`Enter`	Происходит при входе в элемент управления

1.1.6 Элемент управления ***`CheckBox`***

Компонент **`CheckBox`** (кнопка с независимой фиксацией – флажок Windows)  .

Компонент расположен в группе `Common Controls` палитры компонентов. `CheckBox` используется как флажок-переключатель и позволяет пользователю выбрать / отменить определенную опцию, т. е. выдает результат «включен / выключен».

Если компонент `Checkbox` находится в группе себе подобных, то, включив один `Checkbox`, можно включать и остальные, при этом ни один из них не выключится (т. е. не изменит своего состояния). Отличие компонента `Checkbox` от компонента `RadioButton` заключается в том, что одновременно может быть включено несколько `Checkbox'ов`, а компонент `RadioButton` размещенный на форме может быть включен только один и если мы включаем другой компонент `RadioButton`, то первый отключается автоматически.

Этот компонент соответствуют математическим понятиям конъюнкции и дизъюнкции. Когда мы говорим «находится в группе», то имеется в виду, что у множества таких компонентов один родитель (например, одна панель).

1.1.6.1 Свойства элемента управления `CheckBox`

Основные свойства элемента управления `CheckBox` отображены в таблице 1.16.

Таблица 1.16 – Основные свойства `CheckBox`

Название события	Описание события
Appearance	Определяет форму появления компонента (в виде обычного флажка или в виде кнопки)
BackColor	Цвет компонента
BorderColor	Задает цвет обрамления компонента
BorderStyle	Задает стиль границы компонента `CheckBox`
BorderWidth	Задает ширину границы компонента
Checked	По этому свойству в режиме исполнения приложения можно определить, включен или выключен флажок
CheckAlign	Свойство, позволяющее открыть выпадающий список, где можно выбрать схему размещения флажка в поле компонента (его можно разместить в девяти местах окна компонента, но при этом `Appearance` должно быть не кнопкой)
Font	Возвращает свойства шрифта
ForeColor	Задает основной цвет (обычно цвет текста)
Fiatstyle	Определяет стиль появления компонента
Height	Задает высоту элемента управления
Text	Задает текст подписи, связанной с **CheckBox**
TextAlign	Задает выравнивание текста, связанного с **CheckBox**
Visible	Включает / отключает отображение компонента
Width	Задает ширину элемента управления

1.1.6.2 Методы элемента управления `CheckBox`

Основные методы элемента управления `CheckBox` отображены в таблице 1.17.

Таблица 1.17 – Основные методы `CheckBox`

Название события	Описание события
1	2
ApplyStyle	Копирует любой непустой элемент указанного стиля, замещающего все существующие элементы стиля управления
DataBind ()	Связывает источник данных к вызванному серверному элементу управления и всем его дочерним элементам
FindForm	Позволяет объекту освободить ресурсы и выполнить другие операции очистки

Продолжение таблицы 1.17

1	2
Focus	Задает фокус ввода элемента управления
LoadViewState	Загружает предварительно сохраненное состояние `CheckBox` управления
OnCheckedChanged	Повышает `CheckedChanged` событие `CheckBox` управления. Это позволяет обрабатывать событие напрямую
OpenFile	Получает поток, используется для чтения файлов
SaveViewState	Сохраняет изменения в `CheckBox`
ToString	Возвращает строку, которая представляет текущий объект

1.1.6.3 События элемента управления `CheckBox`

Основные события элемента управления `CheckBox` отображены в таблице 1.18.

Таблица 1.18 – Основные события `CheckBox`

Название события	Описание события
CheckedChanged	Возникает при каждом изменении свойства `Check`
Click	Возникает при щелчке элемента управления
DragDrop	Возникает при завершении операции перетаскивания
EnabledChanged	Происходит, когда изменяется включенное состояние этого элемента управления
Enter	Происходит, когда элемент управленя становится активным элементом управления данной формы
FontChanged	Событие возникает, когда в `Control` изменяется значение свойства `Font`
ForeColorChanged	Событие возникает, когда в `Control` изменяется значение свойства `ForeColor`
KeyDown	Происходит в момент первого нажатия клавиши
KeyPress	Возникает, когда этот элемент находится в фокусе и пользователь нажимает и отпускает клавишу
KeyUp	Происходит в момент отпускания клавиши
Leave	Происходит, когда элемент управления перестает быть активным элементом управления данной формы
MouseClick	Возникает при щелчке мышью на элементе
TextChanged	Событие возникает, когда в `Control` изменяется значение свойства `Text`
VisibleChanged	Происходит, когда изменяется видимость этого элемента управления

1.1.7 Элемент управления **`ListBox`**

Компонент находится в списке `All Windows Forms` палитры компонентов. Этот компонент выводит список элементов, из которых пользователь может выбрать как один, так и множество элементов.

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

1.1.7.1 Свойства элемента управления `ListBox`

Основные свойства элемента управления `ListBox` отображены в таблице 1.19.

Таблица 1.19 – Основные свойства `ListBox`

Название события	**Описание события**
1	2
`BackColor`	Фоновый цвет компонента
`BorderStyle`	Задает тип границы, отображаемой вокруг `ListBox`
`ColumnWidth`	Задает ширину колонки списка при многоколоночном списке
`Count`	Находит количество элементов в списке, значение которого всегда на единицу больше индекса последней строки списка, потому что последний отсчитывается от нуля
`DataSource`	Задает источник данных, с помощью которого можно заполнять список
`Font`	Параметры шрифта
`ForeColor`	Цвет фона компонента `ListBox`
`MultiColumn`	Если для свойства `MultiColumn` задано значение `true`, элементы списка отображаются в нескольких столбцах и появляется горизонтальная полоса прокрутки. Это позволяет отобразить больше позиций списка и устраняет необходимость его вертикальной прокрутки для поиска нужной позиции. Если для свойства `MultiColumn` задано значение `false`, элементы списка отображаются в одном столбце и появляется вертикальная полоса прокрутки

Продолжение талицы 1.19

1	2
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
ScrollAlwaysVisible	Если задано значение `true`, полоса прокрутки появляется независимо от числа элементов
SelectedItem	Возвращает выбранный элемент
SelectionMode	Свойство определяет, сколько элементов списка можно выбрать одновременно
TabStop	Указывает, может ли пользователь переключать фокус на этот элемент управления клавишей TAB
Visble	Указывает, отображается ли данный элемент или скрыт

1.1.7.2 Методы элемента управления `ListBox`

Основные методы элемента управления `ListBox` отображены в таблице 1.20.

Таблица 1.20 – Основные методы `ListBox`

Название события	Описание события
CreateGraphics()	Создает графический объект для `listBox1`
FindString()	Позволяет найти в списке позицию, содержащую определенную строку поиска
Measurestring (string,Font)	Метод, который измеряет длину строки в пикселах, выводимую данным шрифтом (разные шрифты принимают на экране разное количество пикселов)

1.1.7.3 События элемента управления `ListBox`

Основные события элемента управления `ListBox` отображены в таблице 1.21.

Таблица 1.21 – Основные события `ListBox`

Название события	Описание события
`Add ()`	Добавить элемент в конец списка
`Insert ()`	Вставить элемент внутрь списка
`Clear ()`	Удалить все элементы из списка (очищает список)
`Click()`	Возникает при щелчке элемента управления
`Remove ()`	Удалить заданный элемент из списка

Извлечь строки из компонента можно так:

```
String ^it= ListBoxl->Items[i]->ToString();
```

где i – номер строки (начинается с нуля).

Обнаружить строку, на которой был щелчок мыши (обработка события `click`), можно так:

```
String^it = this-> listBoxl-> Items [this-> listBoxl-> Select-
edlndex] -> ToString();
```

где `Selectedindex` – индекс выбранной строки.

1.1.8 Элемент управления ***`GroupBox`***

Довольно часто вместо отдельных переключателей используют их групповой контейнер – компонент `GroupBox`, который расположен в группе `All Windows Forms` палитры компонентов.

Вообще `GroupBox` используют, чтобы обеспечить разделение компонентов на различные группы, которые становятся для них родителями, для того чтобы компоненты унаследовали некоторые свойства своих родителей. Обычно так делают, чтобы подразделить форму на несколько функций и дизайн осуществлять удобнее: перемещая только `GroupBox`, мы одновременно перемещаем все компоненты, которые в нем находятся.

Перечень свойств компонента отображается в окне **`Properties`** (никаких новых свойств, по сравнению с ранее встречавшимися, мы тут не находим).

1.1.8.1 Свойства элемента управления `GroupBox`. Основные свойства элемента управления `GroupBox` отображены в таблице 1.22.

Таблица 1.22 – Основные свойства `GroupBox`

Название свойства	Описание свойства
`Anchor`	Определяет грани контейнера, к которому привязан определенный элемент управления
`AutoSize`	Определяет, будет ли размер элемента управления автоматически изменятся в соответствии с размером его содержимого
`BackColor`	Фоновый цвет компонента
`Dock`	Указывает, какие границы элемента управления привязаны к контейнеру
`Enabled`	Указывает, включен ли элемент управления
`Font`	Параметры шрифта
`ForeColor`	Основной цвет для отображения текста в данном элементе управления
`Text`	Задает текст подписи, связанной с **`CheckBox`**
`Visble`	Указывает, отображается ли данный элемент или скрыт

1.1.8.2 Методы элемента управления `GroupBox`

Основные методы элемента управления `GroupBox` отображены в таблице 1.23.

Таблица 1.23 – Основные методы `GroupBox`

Название метода	Описание метода
`Focus`	Задает фокус ввода элемента управления
`Hide`	Скрывает элемент управления от пользователя
`OnClick`	Вызывает событие `Click`
`OnDoubleClick`	Вызывает событие `DoubleClick`
`ResetBackColor`	Сбрасывает свойство `BackColor` в значение по умолчанию
`ResetFont`	Сбрасывает свойство `Font` в значение по умолчанию
`ResetText`	Сбрасывает свойство `Text` в значение по умолчанию
`Select()`	Активирует элемент управления
`SetBoundsCore`	Задает указанные границы данного элемента управления
`SetVisibleCore`	Задает элемент управления в указанном видимом состоянии
`Show`	Отображает элемент управления для пользователя

1.1.8.3 События элемента управления `GroupBox`

Основные события элемента управления `GroupBox` отображены в таблице 1.24.

Таблица 1.24 – Основные события `GroupBox`

Название события	Описание события
`AutoSizeChanged`	Происходит при изменении значения свойства `AutoSize`
`BackColorChanged`	Происходит при изменении значения свойства `BackColor`
`Click`	Происходит при щелчке элемента управления `GroupBox`
`DoubleClick`	Вызывается при двойном щелчке мышью элемента управления `GroupBox`
`Enter`	Происходит при входе в элемент управления
`GotFocus`	Генерируется при получении фокуса элементом управления
`Leave`	Происходит, когда фокус ввода покидает элемент управления
`Move`	Происходит при перемещении элемента управления
`Resize`	Происходит при изменении размеров элемента управления
`SizeChanged`	Генерируется при изменении значения свойства `Size`
`StyleChanged`	Происходит при изменении стиля элемента управления
`SystemColorsChanged`	Происходит при изменении системных цветов
`TextChanged`	Происходит при изменении значения свойства `Text`
`VisibleChanged`	Происходит при изменении значения свойства `Visible`

1.1.9 Элемент управления ***`PictureBox`***

Компонент находится в списке `All Windows Forms` палитры компонентов. Через этот компонент в форму выводится графическое изображение.

Какое изображение надо выводить, указывается в свойстве `Image`. Если нажать кнопку с многоточием в поле этого свойства, то откроется диалоговое окно для выбора объекта в форматах `bmp`, `jpeg`, `icon`, `gif`, `png`.

Можно также загрузить изображение в форму, воспользовавшись свойством `ImageLocation` и методами `Load ()` и `LoadAsync ()`.

Компонент содержит в себе свойства, определяющие, как выводить изображение внутри границ самого этого объекта (в форме `PictureBox` отображается в виде пустого квадрата).

Свойство `Image` – задает изображение, загружаемое в компонент (в поле этого свойства имеется кнопка с многоточием, с помощью которой открывается диалоговое окно для загрузки изображения).

Можно загружать и сохранять изображение также и в режиме исполнения приложения с помощью методов класса `PictureBox`.

Так, например, `Load()` позволяет загружать изображение из файла, путь которому указан в свойстве `imageLocation`. Если в этом свойстве не задавать пути, а указать его в переменной типа `string`, то с помощью метода в форму `Load (Url)` также можно загрузить изображение. В этом случае метод `Load ()` сам назначит свойству `imageLocation` значение переменной url и далее станет работать как этот же метод в своей первой форме (т. е. без параметра).

1.1.9.1 Свойства элемента управления `PictureBox`

Основные свойства элемента управления `PictureBox` отображены в таблице 1.25.

Таблица 1.25 – Основные свойства `PictureBox`

Название свойства	**Описание свойства**
`Anchor`	Определяет грани контейнера, к которому привязан определенный элемент управления
`BackColor`	Фоновый цвет компонента.
`BorderStyle`	Возвращает или задает стиль границы для элемента управления
`Dock`	Указывает, какие границы элемента управления привязаны к контейнеру
`Image`	Изображение, отображаемое в `PictureBox`
`Enabled`	Указывает, включен ли элемент управления
`Font`	Параметры шрифта
`SizeMode`	Определяет, как будет обрабатываться размещение рисунка в данной области рисунка и изменение размеров элемента управления
`Visble`	Указывает, отображается ли данный элемент или скрыт

1.1.9.2 Методы элемента управления `PictureBox`

Основные методы элемента управления `PictureBox` отображены в таблице 1.26.

Таблица 1.26 – Основные методы `PictureBox`

Название метода	Описание метода
`Focus`	Задает фокус ввода элемента управления
`GetScaledBounds`	Задает границы, внутри которых масштабируется элемент управления
`Hide`	Скрывает элемент управления от пользователя
`InvokeOnClick`	Вызывает событие `Click` для указанного элемента управления
`Load()`	Отображает изображение, указанное в свойстве `ImageLocation` объекта `PictureBox`
`Load(String)`	Задает значение свойства `ImageLocation`, равное указанному `URL`-адресу, и отображает указанное изображение
`OnClick`	Вызывает событие `Click`
`OnEnter`	Вызывает событие `Enter`
`OnLeave`	Вызывает событие `Leave`
`OnPaintBackground`	Рисует фон элемента управления
`ProcessKeyMessage`	Обрабатывает сообщение клавиатуры
`Scale(SizeF)`	Масштабирует элемент управления и любые его дочерние элементы с использованием заданного коэффициента масштабирования
`Select()`	Активирует элемент управления
`Show`	Отображает элемент управления для пользователя

1.1.9.3 События элемента управления `PictureBox`

Основные события элемента управления `PictureBox` отображены в таблице 1.27.

Таблица 1.27 – Основные события `PictureBox`

Название события	Описание события
1	2
`BackColorChanged`	– происходит при изменении значения свойства `BackColor`.

Продолжение таблицы 1.27

1	2
Click	Происходит при щелчке элемента управления
FontChanged	Происходит при изменении значения свойства `Font`
ForeColorChanged	Происходит при изменении значения свойства `ForeColor`
GotFocus	Генерируется при получении фокуса элементом управления
Leave	Происходит при потере фокуса ввода объектом `PictureBox`
Resize	Происходит при изменении размеров элемента управления
VisibleChanged	Происходит при изменении значения свойства `Visible`

1.1.10 Элемент управления `Label`

Элементы управления `Windows Forms Label` предназначены для отображения текста или изображений, которые пользователь не может изменить с клавиатуры. Они используются для идентификации объектов в форме, например, для описания, что произойдет с элементом управления после выполнения на нем щелчка мышью, или для отображения сведений в ответ на процесс или событие времени выполнения в приложении. Например, имеется возможность использовать надписи для добавления описательных заголовков в текстовые поля, списки, поля со списком и т. д. Кроме того, возможно написание кода, который изменяет текст, отображаемый в надписи, в ответ на события во время выполнения. Например, если приложению требуется несколько минут на обработку изменения, можно отобразить в надписи сообщение о статусе обработки.

1.1.10.1 Свойства элемента управления `Label`

Основные свойства элемента управления `Label` отображены в таблице 1.28.

Таблица 1.28 – Основные свойства `Label`

Название свойства	**Описание свойства**
1	2
BorderStyle	Возвращает или задает стиль границы для элемента управления

TextAlign	Возвращает или задает выравнивание текста в метке
Text	Получает или задает текст, сопоставленный с этим элементом управления
UseMnemonic	Возвращает или задает значение, показывающее, интерпретируется ли знак амперсанда (&) в свойство Text элемента управления как знак префикса для ключа доступа

Остальные свойства аналогичны ранее рассматриваемым свойствам предыдущих компонентов.

1.1.10.2 Методы элемента управления Label. Методы компонента Label в основном совпадают с методами ранее рассмотренных компонентов.

1.1.10.3 События элемента управления Label. События компонента Label в основном совпадают с событиями ранее рассмотренных компонентов.

1.2 Задание к лабораторной работе

1 Проверить наличие личной папки. При необходимости создать ее.

2 В личной папке создать вложенную папку для лабораторной работы № 1, присвоив этой папке имя ***Visual_Lab1***.

3 Войти в среду **Visual Studio 2010**.

4 После запуска **Microsoft Visual Studio 2010** появляется следующая стартовая страница, которая показана на рис. 1.4.

Следующим шагом является создание нового проекта. Для этого в меню **Файл** необходимо выбрать **Создать → Проект** (или комбинацию клавиш **Ctrl + Shift + N**). Результат выбора пунктов меню для создания нового проекта показан на рис. 1.5.

Среда Visual Studio отобразит окно **Создать Проект**, в котором необходимо выбрать тип создаваемого проекта. Проект используется в Visual Studio для логической группировки нескольких файлов, содержащих исходный код, на одном из поддерживаемых языков программирования, а также любых вспомогательных файлов. Обычно после сборки проекта (которая включает компиляцию всех всходящих в проект файлов исходного кода) создается один исполняемый модуль.

В окне **Создать Проект** следует развернуть узел Visual C++, обратиться к пункту CLR и на центральной панели выбрать *Приложение Windows Form*.

Затем в поле редактора **Имя** (где по умолчанию имеется <Введите имя>) следует ввести имя проекта, например, **Primer**. В поле **Расположение** можно указать путь размещения проекта или выбрать путь размещения проекта с помощью клавиши (кнопки) Обзор. По умолчанию проект сохраняется в специальной папке `Projects`. Пример выбора имени проекта показано на рис. 1.6.

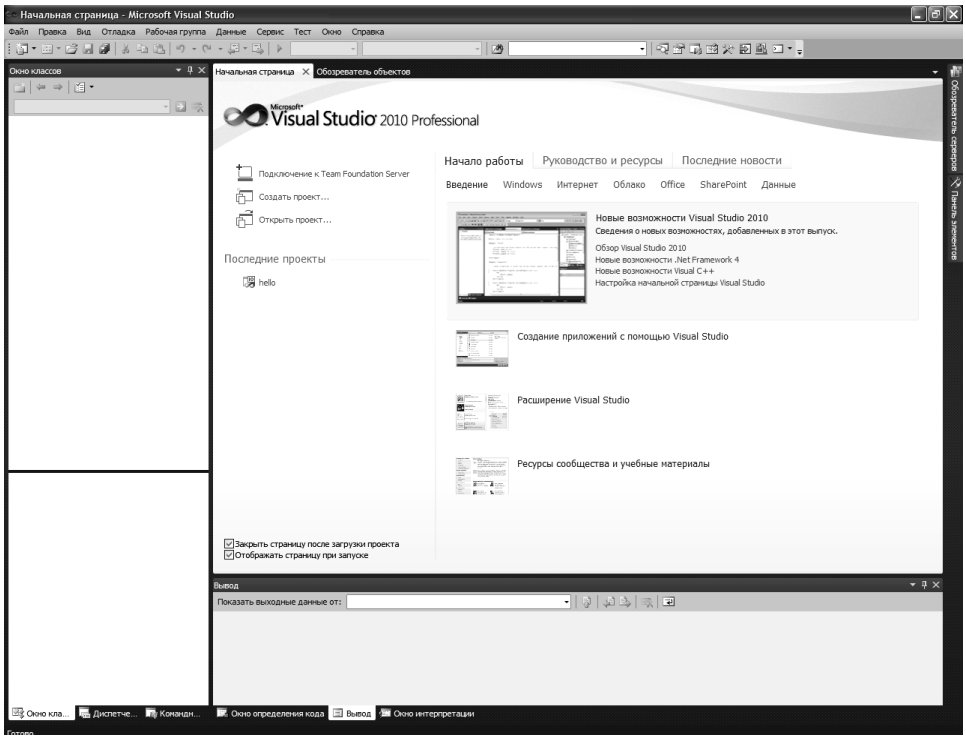


Рисунок 1.4 – Стартовая страница Visual Studio 2010

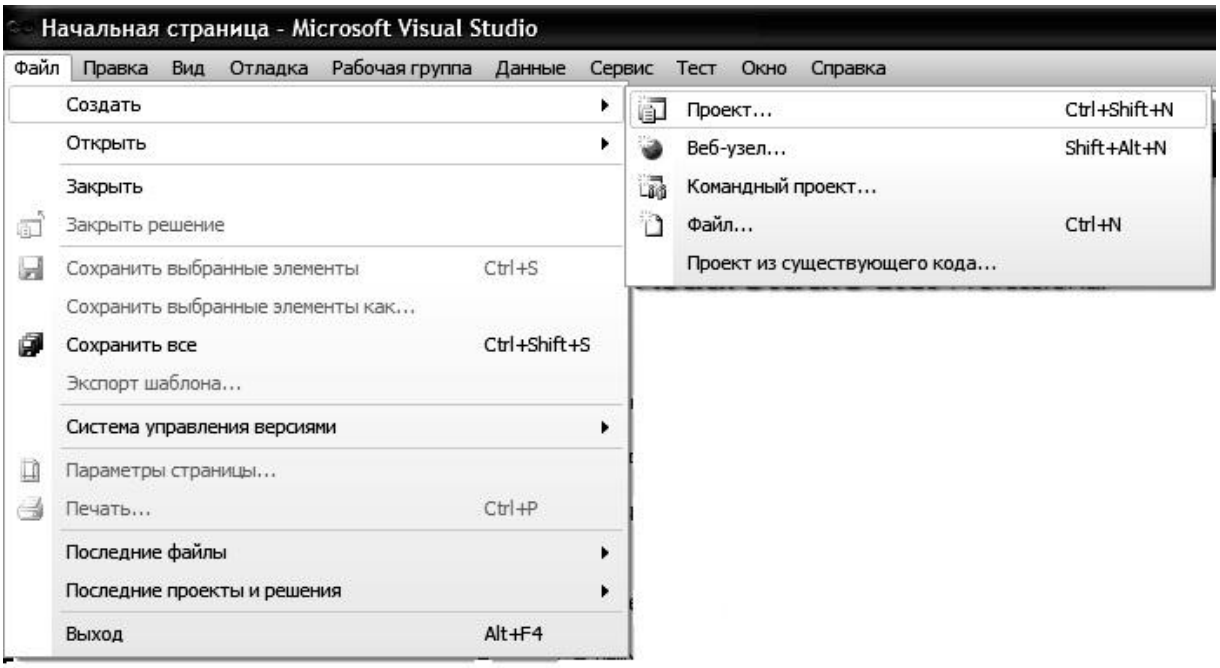


Рисунок 1.5 – Окно с выбором нового проекта

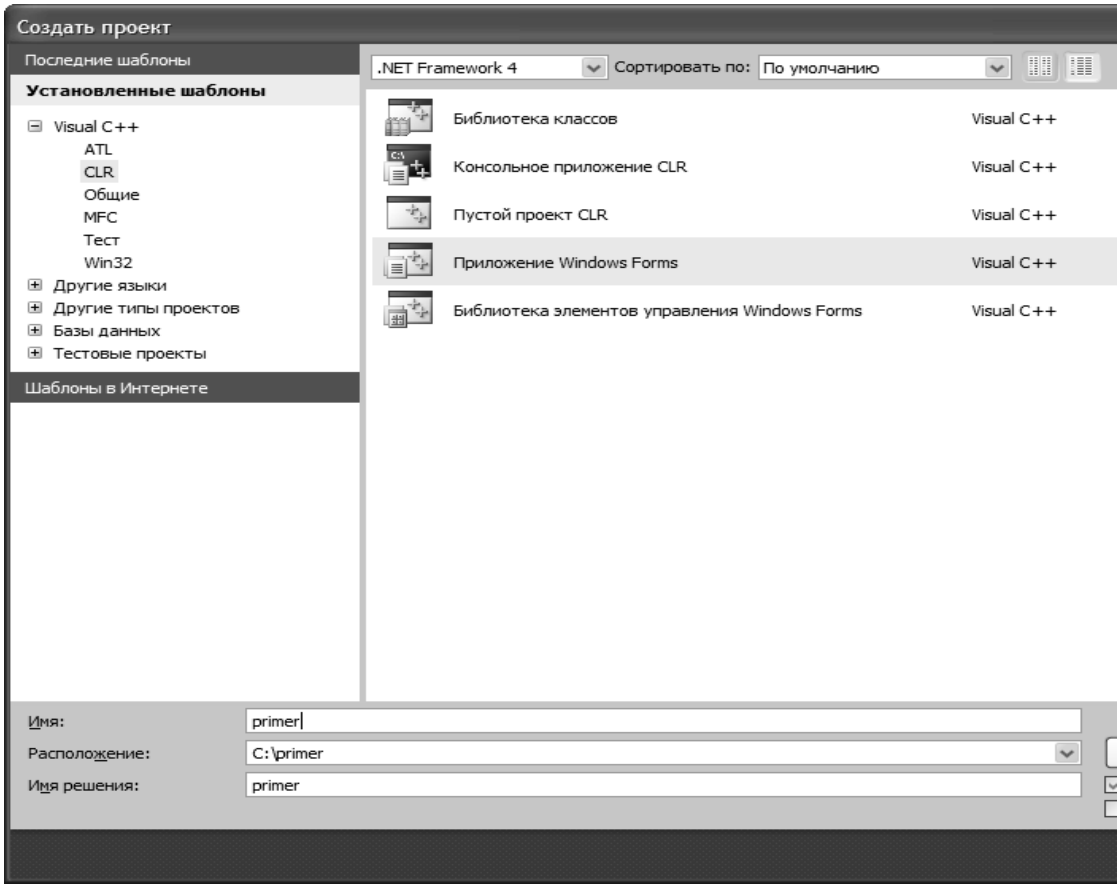


Рисунок 1.6 – Пример задания имени проекта

После нажатия кнопки **OK** откроется окно (рис. 1.7).

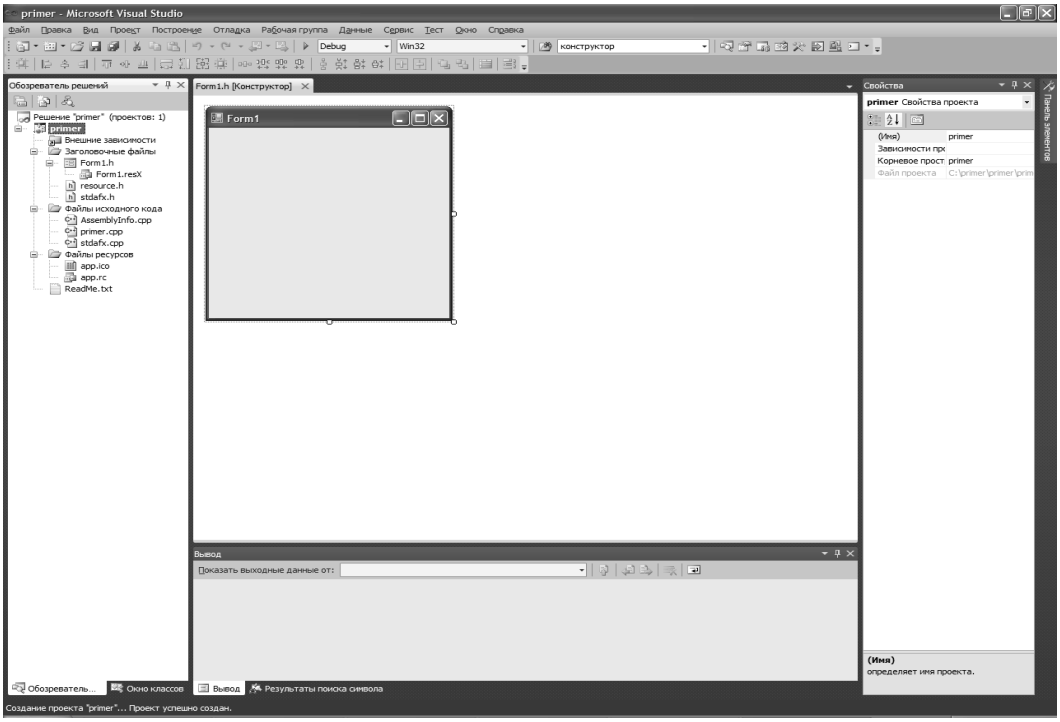


Рисунок 1.7 – Окно, подготовленное к созданию приложения на языке C++

Для запуска программы на ее исполнение из окна редактора в **Visual Studio 2010** можно нажать клавишу **F5.** На рис. 1.8 показан результат исполнения первой программы.

Рисунок 1.8 – Вывод формы программы на языке С

5 Легко убедиться, что появившееся на экране окно-форма обладает всеми свойствами окна **`Windows`**. Его можно перемещать по экрану, изменять размеры, сворачивать в значок и разворачивать на весь экран. Окно имеет стандартные интерфейсные элементы: заголовок, оконное меню, кнопки управления размером и кнопку закрытия. Однако на этом возможности окна и исчерпываются. Это естественно, ведь в проекте еще ничего, кроме формы, не создано.

6 Изменить значение свойства **`Text`** – занести текст, который будет показываться в заголовке формы при выполнении приложения. Например: ***«Лабораторная работа 1. Выполнил Иванов И. И., группа ЭСА-10-1».***

7 Свойству **`FormBorderStyle`** формы (стиль рамки окна) присвоить значение `FixedToolWindow`. Это значение определяет окно как диалоговое, его размеры на этапе прогона приложения (в процессе работы приложения) не могут быть изменены.

8 Вызвать панель элементов, нажав на кнопку  или комбинацию клавиш (Ctrl+Alt+X).

9 Поместить на форму элемент **`Button1`** (командная кнопка) и написать на ней ***«Вывод текста»*** (т. е. занести этот текст в свойство **`Text`**).

10 Аналогично создать кнопку **`Button2`** и написать на ней ***«Запуск формы»*** и кнопку **`Button3`** и написать на ней ***«Выход».*** С помощью свойства **`Font`** можно изменить размер шрифта и написание текста на кнопке. С помощью свойства **`ForeColor`** можно изменить цвет шрифта.

11 Поместить на форму компонент **`listBox1`** (многострочный редактор).

12 Поместить на форму компонент **`GroupBox1`** и в свойстве **`Text`** написать «**Стиль шрифта**». Это компонент-контейнер. Аналогично, как и для кнопок, с помощью свойств **`Font`** и **`ForeColor`** можно изменить размер шрифта, написание текста и цвет шрифта.

13 Разместить в контейнере **`GroupBox1`** четыре компонента типа ***`TcheckBox`*: `CheckBox1`**, … , **`CheckBox4`**, подписав их в свойстве **`Text`** как: «***Полужирный***», «***Курсивный***», «***Подчеркнутый***» и «***Перечеркнутый***».

14 Поместить на форму компонент **`TextBox1`**.

15 Слева от компонента **`TextBox1`** поместить компонент **`Label1`** с подписью (свойство **`Text`**) «***Введите текст***». Также с помощью свойств **`Font`** и **`ForeColor`** можно изменить размер шрифта, написание текста и цвет шрифта.

16 Поместить на форму компонент **`GroupBox2`** и в свойстве **`Text`** этого компонента написать «***Цвет шрифта***». Это компонент-контейнер.

17 Разместить в контейнере **`GroupBox2`** пять компонентов-радиокнопок: **`RadioButton1`**, …, **`RadioButton5`**, подписав их как «***Красный***», «***Синий***», «***Зеленый***», «***Желтый***» и «***Черный***». В соответствии с этими подписями установить цвет в свойстве **`BackColor`** (цвет фона) каждого из этих компонентов. Чтобы на этом фоне смотрелись надписи, для компонентов зеленого и черного цвета изменить цвет символов свойство **`ForeColor`** на белый.

18 Используя команду **Проект ⇒ Добавить новый элемент**, создать новую форму **`Form2`,** определив ее имя (рис. 1.9).

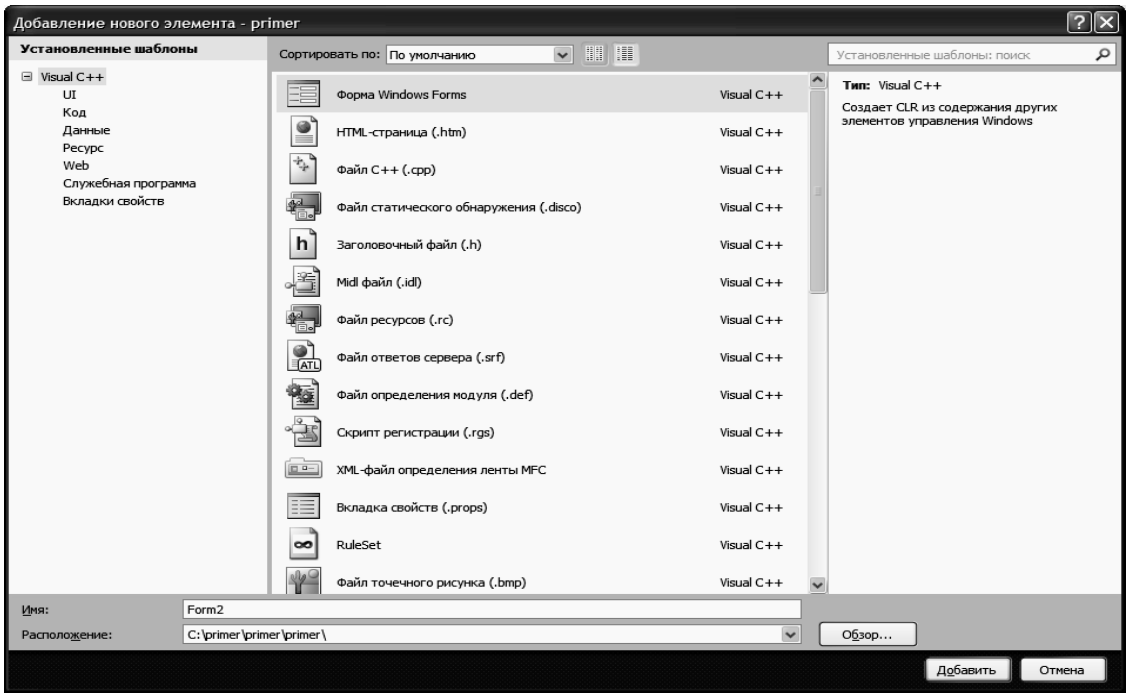


Рисунок 1.9 – Добавление новой формы в проект

19 В свойство **`Text`** этого компонента занести текст «***О программе***». Свойству **`FormBorderStyle`** формы (стиль рамки окна) присвоить значение ***`FixedToolWindow`***. Цветовое оформление выбрать по своему усмотрению (свойство **`BackColor`**) (рис. 1.10; 1.11).

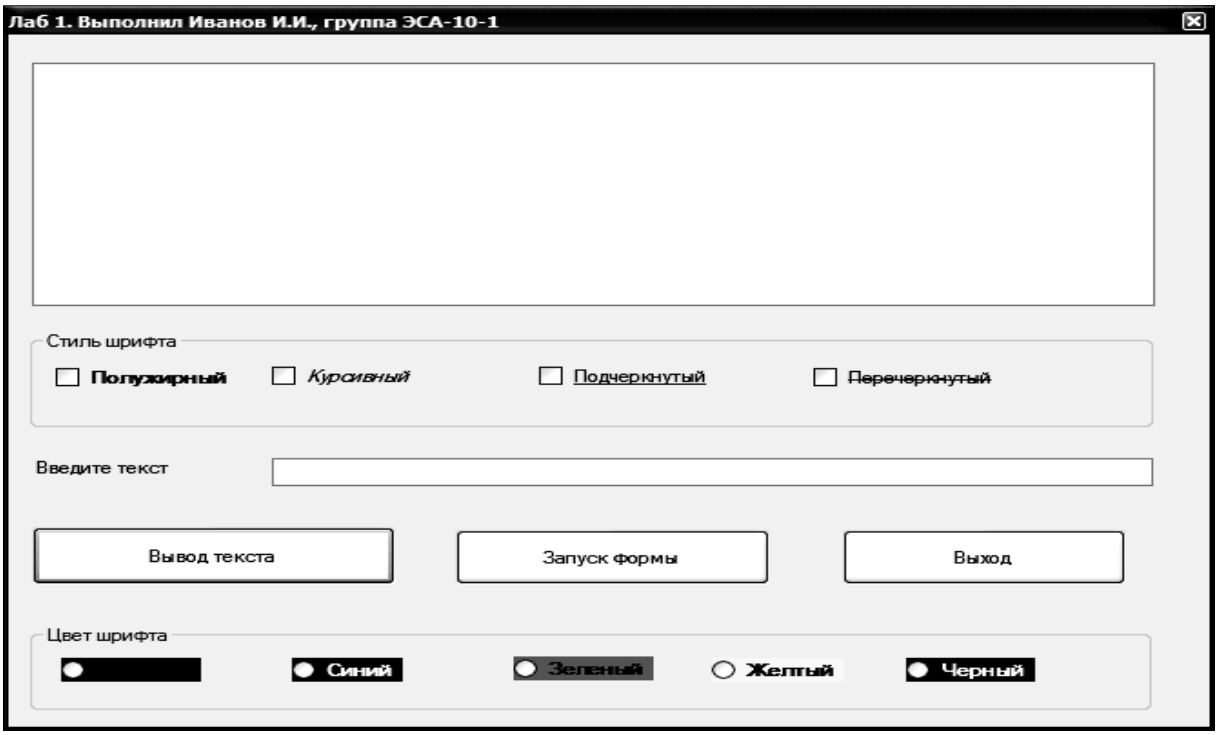


Рисунок 1.10 – Вид Form1 лабораторной работы

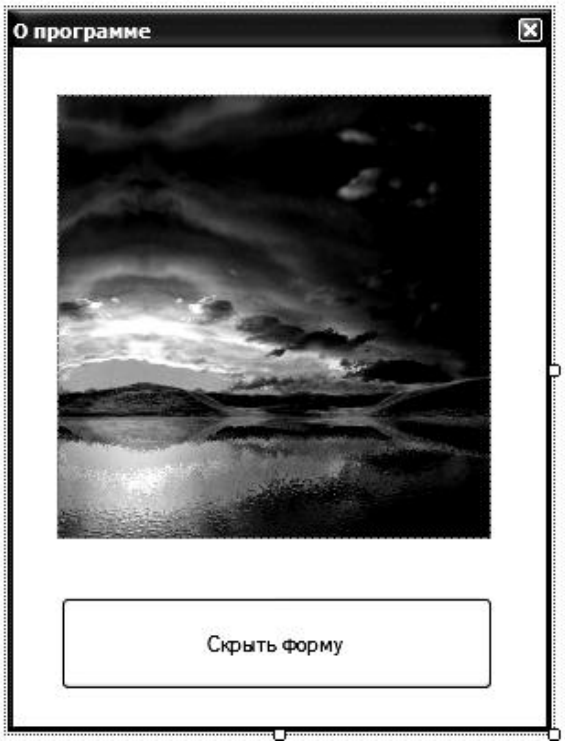


Рисунок 1.11 – Вид Form2 лабораторной работы

20 Поместить на новую форму компонент **Button1** (кнопка). На кнопке написать «***Скрыть форму***».

21 Поместить на форму **Form2** компонент **PictureBox1** (отображение картинок) и с помощью свойства **Image** загрузить в него соответствующий тематике рисунок. С помощью свойства **SizeMode** вписываем нашу картинку в блок картинки, выбрав пункт **StretchImage.**

22 В результате проведенных операций должно получиться две формы примерно такого вида, как показано на рис. 1.10; 1.11 (показан вид, который формы будут иметь на этапе выполнения).

Формы не должны обязательно иметь представленный выше вид, но общая идея лабораторной работы, т. е. знакомство с основными компонентами и их свойствами, должна быть соблюдена.

23 Запустить приложение на выполнение. Несмотря на наличие в форме всевозможных элементов управления, они не выполняют никаких действий. Например, можно выбрать любой стиль шрифта или цвет, но этот выбор нигде не отразится. Это естественно, т. к. в проекте еще не разработан ни один обработчик события, поэтому ни одно событие не обрабатывается, т. е. приложение не реагирует еще ни на одно событие. Форма **Form2** вообще никогда не сможет появиться на экране, несмотря на то, что она зарегистрирована в проекте. Наличие командной кнопки «***Запуск формы»*** еще не обеспечивает ее показ на экране. Необходим соответствующий программный код, который бы в ответ на щелчок на этой кнопке показывал форму.

24 Сделать первую форму активной. Щелчком на компоненте **Button2** (командная кнопка ***«Запуск формы»***) выбрать (активизировать) его. В окне *Свойства* отображаются свойства и события выбранного компонента, т. е. при активизации **Button2** будут показаны свойства и события именно этого объекта. Перейти на вкладку **События**  **.** Найти строку с именем события ***Click*** (Щелчок)**.** Двойной щелчок в правой колонке этой строки (события) переводит в окно ввода кода обработчика этого события. Для объекта типа ***Button*** событие ***button2_Click*** является событием по умолчанию, код которого раскрывается двойным щелчком на объекте. Если обработчик события еще не разработан, ***Visual Studio 2010*** создает заготовку для его создания. Создать следующую процедуру обработки этого события (эта процедура показывает форму **Form2**, делая ее видимой). Для этого добавим строку

```
#include "Form2.h"
```

ссылку на **Form2** сразу после первой строки в коде программы **Form1.**

```
private: System::Void button2_Click(System::Object^  sender,
System::EventArgs^  e) {
                Form2^ f = gcnew Form2();
            this->Hide();
            f->ShowDialog();
            this->Show();
}
```

24 Сделать активной вторую форму (**Form2**). В окне ***Свойства*** перейти на вкладку **События.** Из списка компонентов, установленных на форме, выбрать **Button1**. Найти событие ***Click.*** Двойным щелчком по этому событию перейти в окно ввода кода. Создать процедуру обработки этого события, закрывающую (делающую невидимой) форму **Form2**:

```
    private: System::Void button1_Click(System::Object^
sender, System::EventArgs^  e) {this->Hide();};
```

25 Запустить приложение на выполнение. Теперь при щелчке на командной кнопке **Button2** (***«Запуск формы»***) на экране появится форма **Form2**, т. к. событие Щелчок на этой кнопке будет обработано и этот обработчик покажет форму (сделает ее видимой). Аналогично, щелчок на соответствующей кнопке формы **Form2** уберет с экрана эту форму (сделает ее невидимой). Закрыть приложение.

26 Сделать окно первой формы активным**.** Щелчком на компоненте **Button1** (кнопка ***«Вывод текста»***) выбрать (активизировать) его. Двойной щелчок левой кнопки мышки по этому элементу управления переводит в окно ввода кода обработчика этого события.

```
private: System::Void button1_Click(System::Object^  sender,
System::EventArgs^  e)
{ между этими скобками вставит код }
```

и в заготовку подпрограммы между фигурными скобками вставить следующий код:

```
if (textBox1->Text!=""){this ->listBox1->Items->Clear();
this->listBox1->Font = (gcnew System::Drawing::Font(L"Arial",
14, System::Drawing::FontStyle::Regular, System:
:Drawing::GraphicsUnit::Point, static_cast<System:
:Byte>(204))); this->listBox1->ForeColor = System:
:Drawing::Color: :Black; this ->listBox1->Items->Add(this -
>textBox1->Text);
//Устанавливает красный цвет для текста
if (radioButton1->Checked == true){ this->listBox1->ForeColor
= System::Drawing::Color::Red;}
//Устанавливает синий цвет для текста
if (radioButton2->Checked == true){ this->listBox1->ForeColor
= System::Drawing::Color::Blue;}
//Устанавливает зеленый цвет для текста
if (radioButton3->Checked == true){ this->listBox1->ForeColor
= System::Drawing::Color::Green;}
//Устанавливает желтый цвет для текста
if (radioButton4->Checked == true){ this->listBox1->ForeColor
= System::Drawing::Color::Yellow;}
//Устанавливает черный цвет для текста
if (radioButton5->Checked == true){ this->listBox1->ForeColor
= System::Drawing::Color::Black;}
```

```
//Устанавливает атрибут «жирность» для текста
if (checkBox1->Checked == true){this->listBox1->Font = (gcnew
System::Drawing::Font(L"Arial", 14, System::Drawing:
:FontStyle:: Bold, System::Drawing::GraphicsUnit::Point,
     static_cast<System::Byte>(204)));}
//Устанавливает атрибут «курсив» для текста
if (checkBox2->Checked == true){this->listBox1->Font = (gcnew
System::Drawing::Font(L"Arial", 14, System::Drawing:
:FontStyle:: Italic, System::Drawing::GraphicsUnit::Point,
     static_cast<System::Byte>(204)));}
//Устанавливает атрибут «подчеркнутый» для текста
if (checkBox3->Checked == true){this->listBox1->Font = (gcnew
System::Drawing::Font(L"Arial", 14, System::Drawing:
:FontStyle:: Underline, System::Drawing: :GraphicsUnit::Point,
static_cast<System::Byte>(204)));}
//Устанавливает атрибут «зачеркнутый» для текста
if (checkBox4->Checked == true){this->listBox1->Font = (gcnew
System::Drawing::Font(L"Arial", 14, System::Drawing:
:FontStyle:: Strikeout, System::Drawing::GraphicsUnit::Point,
static_cast<System::Byte>(204)));}
//Устанавливает атрибуты «жирность», «курсив» для текста
if ((checkBox1->Checked == true)&&(checkBox2->Checked ==
true)){this->listBox1->Font = (gcnew System::Drawing:
:Font(L"Arial", 14, static_cast<System:
:Drawing::FontStyle>((System::Drawing::FontStyle::Bold |
System::Drawing::FontStyle::Italic)), System::Drawing:
:GraphicsUnit::Point, static_cast<System::Byte>(204)));}
//Устанавливает атрибуты «жирность», «курсив» и «подчеркнутый» для текста
if ((checkBox1->Checked == true)&&((checkBox2->Checked ==
true))&&(checkBox3->Checked == true))this->listBox1->Font =
(gcnew System::Drawing::Font(L"Arial", 14,
static_cast<System::Drawing:
:FontStyle>(((System::Drawing::FontStyle::Bold | System:
:Drawing::FontStyle::Italic)| System::Drawing::FontStyle:
:Underline)), System::Drawing::GraphicsUnit::Point,
static_cast<System: :Byte>(204)));
//Устанавливает атрибуты «жирность», «курсив», «подчеркнутый» и «зачеркнутый»
для текста
if ((checkBox1->Checked == true)&&(checkBox2->Checked ==
true)&&(checkBox3->Checked == true)&&(checkBox4->Checked ==
true))this->listBox1->Font = (gcnew System::Drawing:
:Font(L"Arial", 14, static_cast<System::Drawing:
:FontStyle>(System::Drawing::FontStyle::Bold | System:
:Drawing::FontStyle::Italic | System::Drawing:
:FontStyle::Underline | System::Drawing:
:FontStyle::Strikeout), System::Drawing::GraphicsUnit::Point,
static_cast<System::Byte>(204)));
//Устанавливает атрибуты  «курсив», «подчеркнутый» и «зачеркнутый» для текста
if ((checkBox2->Checked == true)&&(checkBox3->Checked ==
true)&&(checkBox4->Checked == true))this->listBox1->Font =
(gcnew System::Drawing::Font(L"Arial", 14, static_cast<System:
:Drawing::FontStyle>(System::Drawing::FontStyle::Italic |
```

```
System::Drawing::FontStyle::Underline | System:
:Drawing::FontStyle::Strikeout), System::Drawing:
:GraphicsUnit::Point, static_cast<System::Byte>(204)));
```
//Устанавливает атрибуты «курсив», «подчеркнутый» для текста
```
if ((checkBox2->Checked == true)&&(checkBox3->Checked ==
true))this->listBox1->Font = (gcnew System::Drawing:
:Font(L"Arial", 14, static_cast<System::Drawing:
:FontStyle>(System::Drawing::FontStyle::Italic | System:
:Drawing::FontStyle::Underline), System::Drawing:
:GraphicsUnit::Point, static_cast<System::Byte>(204)));
```
//Устанавливает атрибуты «курсив», «зачеркнутый» для текста
```
if ((checkBox2->Checked == true)&&(checkBox4->Checked ==
true))this->listBox1->Font = (gcnew System::Drawing:
:Font(L"Arial", 14, static_cast<System:
:Drawing::FontStyle>(System::Drawing::FontStyle::Italic |
System::Drawing::FontStyle::Strikeout), System::Drawing:
:GraphicsUnit::Point, static_cast<System::Byte>(204)));
```
//Устанавливает атрибуты «жирность», «подчеркнутый» для текста
```
if ((checkBox1->Checked == true)&&(checkBox3->Checked ==
true))this->listBox1->Font = (gcnew System::Drawing:
:Font(L"Arial", 14, static_cast<System::Drawing:
:FontStyle>(System::Drawing::FontStyle::Bold | System:
:Drawing::FontStyle::Underline), System::Drawing:
:GraphicsUnit::Point, static_cast<System::Byte>(204)));
```
//Устанавливает атрибуты «подчеркнутый», «зачеркнутый» для текста
```
if ((checkBox3->Checked == true)&&(checkBox4->Checked ==
true))this->listBox1->Font = (gcnew System::Drawing:
:Font(L"Arial", 14, static_cast<System::Drawing:
:FontStyle>(System::Drawing::FontStyle::Underline |
System::Drawing::FontStyle::Strikeout), System::Drawing:
:GraphicsUnit::Point, static_cast<System::Byte>(204)));
```
//Устанавливает атрибуты «жирность», «зачеркнутый» для текста
```
if ((checkBox1->Checked == true)&&(checkBox4->Checked ==
true))this->listBox1->Font = (gcnew System::Drawing:
:Font(L"Arial", 14, static_cast<System::Drawing:
:FontStyle>(System::Drawing::FontStyle::Bold |
System::Drawing::FontStyle::Strikeout), System::Drawing:
:GraphicsUnit::Point, static_cast<System::Byte>(204)));
```
//Устанавливает атрибуты «жирность», «подчеркнутый», «зачеркнутый» для текста
```
if ((checkBox1->Checked == true)&&(checkBox3->Checked ==
true))this->listBox1->Font = (gcnew System::Drawing:
:Font(L"Arial", 14, static_cast<System::Drawing:
:FontStyle>(System::Drawing::FontStyle::Bold | System:
:Drawing::FontStyle::Underline), System::Drawing:
:GraphicsUnit::Point, static_cast<System::Byte>(204)));
```
//Устанавливает атрибуты «жирность», «курсив», «зачеркнутый» для текста
```
if ((checkBox1->Checked == true)&&((checkBox2->Checked ==
true))&&(checkBox4->Checked == true))this->listBox1->Font =
(gcnew System::Drawing::Font(L"Arial", 14, static_cast<System:
:Drawing::FontStyle>((System::Drawing::FontStyle::Bold
|System::Drawing::FontStyle::Italic|System::Drawing::FontStyle
::Strikeout)), System::Drawing::GraphicsUnit::Point,
static_cast<System::Byte>(204)));
```

```
          }
else
{MessageBox::Show( "Заполните пожалуйста данные", "Ошибка ввода данных",
     MessageBoxButtons::OK, MessageBoxIcon::Exclamation );}
```

27 Сделать окно первой формы активным**.** Щелчком на компоненте **`Button3`** (командная кнопка ***«Выход»***) выбрать (активизировать) его. В окне ***Свойства*** перейти на вкладку **События.** Найти строку с именем события ***`Click`*** (Щелчок)**.** Двойной щелчок в правой колонке этой строки (события) переводит в окно ввода кода обработчика этого события. Создать следующую процедуру обработки события:

```
     private: System::Void button3_Click(System::Object^  sender, System::EventArgs^  e)
          { Application::Exit();}
```

28 Нажав клавишу **F5**, запустить приложение на выполнение. В случае необходимости выполнить отладку. Объяснить логику функционирования приложения.

29 По результатам работы составить отчет.

1.3 Контрольные вопросы

1. Как создать проект `Windows Forms` и его сохранение?
2. Как создать вторую форму и ее вызов из первой формы?
3. Компонент `Forms` и его основные свойства.
4. Компонент `Memo` и его основные свойства.
5. Компонент `Label` и его основные свойства.
6. Компонент `TextBox` и его основные свойства.
7. Компонент `Button` и его основные свойства.
8. Компонент `ListView` и его основные свойства.
9. Компонент `RadioButton` и его основные свойства.
10. Компонент `ChecBox` и его основные свойства.
11. Компонент `GroupBox` и его основные свойства.
12. Компонент `ListBox` и его основные свойства.
13. Компонент `PictureBox` и его основные свойства.

2 ЛАБОРАТОРНАЯ РАБОТА 2. УСЛОВНЫЕ ОПЕРАТОРЫ. ВЫЧИСЛЕНИЕ ЗНАЧЕНИЯ ФУНКЦИИ, ЗАДАННОЙ УСЛОВНО

Цель: Создание простого законченного Windows приложения, включающего ввод исходных данных, расчет и вывод результатов расчета на форму. Знакомство с возможностями ***Visual Studio 2010***. Проектирование разветвляющегося вычислительного процесса.

2.1 Теоретические сведения

В языке программирования **C++** используются несколько конструкций для принятия решений:

- оператор `if`;
- оператор `switch`;
- условный оператор `?` (оператор условия).

Для прерывания программного цикла при некотором условии применяется утверждение (оператор) `break`, для продолжения итераций цикла при выполнении некоторых условий применяется утверждение (оператор) `continue`, для выхода из функции при выполнении некоторых условий применяется оператор `return`, для перехода к заданному месту программы применяется оператор `goto`, хотя считается, что в программировании не существует ситуаций, в которых нельзя обойтись без оператора `goto`. Утверждение `break` применяется также в теле оператора `switch`.

2.1.1 Условный оператор `if` в языке программирования C++

Общая форма записи оператора `if`:

```
if (условие) {действие};
```

В операторе `if` используется результат вычисления условия, заключенного в круглые скобки, на основе которого принимается решение. Результат вычисления условия (действие) может быть арифметическим или логическим. Если результат выполнения условия будет истинным, то возможно выполнить несколько действий. Для этого следует использовать фигурные скобки, например:

```
if (условие)
{
действие1;
действие2;
...
}
```

2.1.2 Конструкция `if-else`

Общая форма записи конструкции `if-else`:

```
if (условие)действие1;
else
действие2;
```

Если выполняется условие, то будет выполняться фрагмент программы `действие1`, в противном случае будет выполняться `действие2`.

Каждое из утверждений может быть множественным. В таком случае применяются фигурные скобки:

```
if (условие)
{
действие1;
действие2;
...
}
else
{
действие3;
действие4;
...
}
```

2.1.3 Конструкция `if-else if-else if-...-else`

Форма записи конструкции `if-else if-else if-...-else`:

```
if (условие1)
действие1;

else if (условие2)
действие2;

else if (условие3)
действие3;
...
else
действие;
```

Приведенная конструкция используется для выбора возможных ситуаций, когда проверяются условия `условие1`, `условие2`, `условие3`, ... Соответственно будут выполняться действия: `действие1`, `действие2`, `действие3` и т. д. В случае, когда ни одно из условий не выполняется, выполняются действия, прописанные после оператора `else`.

В случае выполнения множественных действий применяются фигурные скобки для каждого из утверждений:

```
if (условие1)
{
действие1;
...
}

else if (условие2)
{
действие2;
...
}

else if (условие3)
{
действие3;
...
}
...

else
{
действие;
...
}
```

2.1.4 Оператор `switch`

Общая форма записи оператора `switch`:

```
switch (условие) {
case значение1:
действие;
...
break;

case значение2:
действие;
...
break;
...

case значениеn:
действие;
...
break;

default:
действие;
...
break;
}
```

Выражение заключенного в круглые скобки оператора последовательно сравнивается со значениями: `значение 1`, `значение 2`, ..., `значение n`, которые должны быть простыми константами или константными выражениями. В том случае, когда одно из этих значений равно значению, выполняются утверждения, которые следуют за данным значением.

Утверждение `break` сигнализирует об окончании выполнения утверждений и приводит к выходу из оператора `switch`. Утверждение `break` ставится в конце каждого варианта выбора. Если этого не сделать, то выполнение последовательности утверждений перейдет в следующий вариант выбора и будет выполняться до тех пор, пока не встретится утверждение `break`.

Специальный дополнительный вариант `default` будет выполнен в том случае, когда не будет найдено ни одного совпадения.

Операторы `if` и `switch` той или иной синтаксической конструкции существуют практически во всех языках программирования (в первую очередь языках высокого уровня), и их часто называют операторами ветвления.

2.1.5 Условный оператор

В отличие от других операторов языка **C++**, которые могут быть унарными или бинарными, специфический оператор условия является тернарным оператором. Это означает, что у него может быть три операнда.

Общий формат записи оператора условия:

```
условие ? выражение_1 : выражение_2
```

Если в результате вычисления **условия** будет получено значение `TRUE` (истина, не нуль), то выполняется выражение_1 и результатом выполнения оператора условия будет значение, полученное при вычислении этого выражения. Если в результате вычисления условия будет получено значение `FALSE` (ложь, т. е. нуль), то выполняется выражение_2 и результатом выполнения оператора условия будет значение, полученное при вычислении выражение_2.

Оператор условия часто описывают как оператор `?`. Тернарный оператор условия `?` наиболее часто используется для присвоения переменной одного из двух значений в зависимости от некоторого условия.

2.1.6 Оператор `break` (от английского – прерывать)

Оператор или утверждение `break` служит для немедленного выхода из цикла, будь то `while`, `for` или `do-while`. После выхода из цикла выполнение программы продолжается с утверждения (фрагмента программы), непосредственно следующего за циклом.

Если оператор `break` встречается во вложенном цикле (вложенных циклах), то будет прекращено выполнение того цикла, в котором этот оператор встретился.

Необходимость в использовании оператора прерывания `break` в теле цикла возникает тогда, когда условие продолжения итераций нужно проверять не в начале цикла (как в циклах `while` и `for`) и не в конце тела цикла (как в цикле `do-while`), а в середине тела цикла.

Формат записи оператора `break`:

```
break;
```

2.1.7 Оператор `continue` (от английского – продолжать)

Оператор или утверждение `continue` служит для перехода к следующей итерации цикла.

Оператор `continue` противоположен по действию оператору `break`. Оператор `continue` позволяет в любой точке тела цикла (`while`, `for` или `do-while`) прервать текущую итерацию и перейти к проверке условий продолжения цикла. В соответствии с результатами проверки либо заканчивается выполнение цикла, либо начинается новая итерация. При этом все утверждения (фрагменты программы), которые следуют за оператором `continue` (ключевым словом), автоматически пропускаются.

Формат записи оператора `continue`:

```
continue;
```

2.1.8 Оператор `goto`

Сейчас во многих языках программирования оператор безусловного перехода типа `goto` не используется. Однако в языке программирования **C++** он имеет место. Применение оператора `goto` не является хорошим стилем программирования. Но в некоторых случаях его применение бывает уместно. Иногда, при умелом использовании, оператор `goto` может оказаться весьма полезным, например, если нужно покинуть глубоко вложенные циклы.

Для оператора `goto` всегда необходима метка. Метка – это идентификатор с последующим двоеточием. Метка должна находиться в той же функции, что и оператор `goto`, переход в другую функцию невозможен.

Общий формат записи оператора `goto`:

```
goto метка;
…
метка: заданные действия.
```

Метка может находиться как до, так и после оператора `goto`. С помощью оператора `goto` можно не только выходить из цикла, но и организовать цикл.

Логические операторы отношения приведены в табл. 2.1.

Таблица 2.1 – Логические операторы

№ п/п	Оператор	Операция
1	&&	И
2	||	ИЛИ
3	!	НЕ, отрицание

Ниже приведены операции отношений в убывающей последовательности приоритетов:

Наивысший !
> >= < <=
== !=
&&
Низший ||

Как и в арифметических выражениях, для изменения порядка выполнения операций сравнения и логических операций можно использовать круглые скобки.

Операторы отношения перечислены в табл. 2.2.

Таблица 2.2 – Операторы отношения языка программирования C++

№ п/п	Оператор	Значение
1	==	Равно
2	!=	Не равно
3	<	Меньше
4	<=	Меньше или равно
5	>	Больше
6	>=	Больше или равно

Результат любой операции сравнения или логической операции есть **0** (нуль) или **1**.

Решение практически любой задачи можно запрограммировать самостоятельно от начала до конца. Однако при составлении программ очень часто возникает потребность выполния какие-либо действий, которые уже использовались в различных программах. Например, при математических расчетах нужно вычисление тригонометрических функций, а программированием

этих вычислений уже не раз занималось множество программистов. Поэтому в **Visual C++** входит обширный набор стандартных модулей, содержащих стандартные функции. Такие модули представляют собой готовый откомпилированный и оптимизированный код, предназначенный для решения самых разных задач.

В таблице 2.3 приведен перечень *некоторых* встроенных математических функций **Visual C++** 2010. Прототипы этих функций определены в заголовочном файле math.h. Поэтому, чтобы иметь возможность их использования, необходимо в вызывающем модуле использовать директиву:

#include <math.h>.

Все функции **C++** (не только стандартные) записываются так: сначала следует название функции, потом в круглых скобках – список параметров через запятую (если параметров несколько).

2.1.9 Математические функции в языке программирования C++

Таблица 2.3 – Запись математических функций в C++

Математическая функция	Название функции	Функция на языке программирования C++	Поясненне
1	2	3	4
arccos *x*	арккосинус	#include <math.h> double acos(double x);	аргумент для acos должен находиться в отрезке [–1; 1]. Acos и acosf возвращают значения в радианах на промежутке от 0 до pi
arccosh *x*	обратный гиперболический косинус	#include <math.h> double acosh(double x);	x должен быть больше либо равен 1
arcsin *x*	арксинус	#include <math.h> double asin(double x);	аргумент для asin должен находиться в отрезке [–1; 1]. Asin и asinf возвращают значения в радианах в промежутке от –pi/2 до pi/2.
arcsinh *x*	обратный гиперболический синус	#include <math.h> double asinh(double x);	ни atanh, ни atanhf не являются ANSI C – функциями
arctg *x*	арктангенс	#include <math.h> double atan(double x);	atan и atanf возвращают значения в радианах на промежутке от –pi/2 до pi/2.
arctgh *x*	обратный гиперболический тангенс	#include <math.h> double atanh(double x);	ни atanh, ни atanhf не являются ANSI C – функциями.
$\sqrt[3]{x}$	кубический корень	#include <math.h> double pow(x,1.0/3.0);	Является стандартной функцией ANSI C

Продолжение таблицы 2.3

1	2	3	4
$\cosh x$	гиперболический косинус	#include <math.h> double cosh(double x);	Углы определены в радианах.
e^x	экспонента числа	#include <math.h> double exp(double x);	Является стандартной функцией ANSI C
$\|x\|$	модуль числа (абсолютная величина)	#include <math.h> double fabs(double x);	Является стандартной функцией ANSI C
$\ln x$	натуральный логарифм	#include <math.h> double log(double x);	Является стандартной функцией ANSI C
$\lg x$	логарифм по основанию 10	#include <math.h> double log10(double x);	log10 возвращает значение логарифма по основанию 10 от x. Он определяется как ln(x)/ln(10)
x^y	возведение основания ***x*** в степень ***y***	#include <math.h> double pow(x, y);	Является стандартной функцией ANSI C
$\sqrt{x}$	квадратный корень из числа	#include <math.h> double sqrt(double x);	вычисляет арифметический (неотрицательный) квадратный корень из аргумента
$\sin x$	синус	#include <math.h> double sin(double x);	Углы определены в радианах
$\cos x$	косинус	#include <math.h> double cos(double x);	Углы определены в радианах
$\sinh x$	гиперболический синус	#include <math.h> double sinh(double x);	Углы определены в радианах
$\mathrm{tg}\, x$	тангенс	#include <math.h> double tan(double x);	Углы определены в радианах
$\mathrm{tgh}\, x$	гиперболический тангенс	#include <math.h> double tanh(double x);	Углы определены в радианах. tanh(x) определяется как sinh(x)/cos(x)

2.2 Пример выполнения работы

Создать приложение для вычисления значения функции:

$$y=\begin{cases}\sin(x^2+ax), & \text{если} \quad x\le 0\\ 1-\dfrac{1+\sqrt{(x^2+ax)}}{e^{\sin(x)}(1+x)}, & \text{если} \quad 0<x\le a\\ \dfrac{\cos(x^2-a^2)}{\sqrt{1-\sin(a-x)}}-\dfrac{1-\sin(a-x)}{e^{\sin(x)}}, & \text{если} \quad x>a\end{cases}.$$

1 Войти в среду ***Visual Studio 2010.***

2 В окне **Создать Проект** следует развернуть узел `Visual C++`, обратиться к пункту `CLR` и на центральной панели выбрать *Приложение* `Windows Form`.

3 Затем в поле редактора **Имя** (где по умолчанию имеется <Введите имя>) следует ввести имя проекта, **`Visual_Lab2`**. В поле **Расположение** можно указать путь размещения проекта или выбрать путь размещения проекта с помощью клавиши (кнопки) **Обзор** (например N:\\CI\2_trim\Lab2).

4 Для формы изменить значение свойства **`Text`,** занеся, напрммер, следующие данные: ***«Выполнил студент группы ЭСА-11-1 Иванов П. А. Лабораторная работа 2».***

5 Свойству **`FormBorderStyle`** формы (стиль рамки окна) присвоить значение ***`FixedToolWindow`***. Это значение определяет окно как диалоговое, его размеры на этапе прогона приложения (в процессе работы приложения) не могут быть изменены.

6 Поместить на форму компонент **`Button1`** (командная кнопка) и написать на ней ***«Расчет»*** (т. е. занести этот текст в свойство **`Text`**).

7 Аналогично создать кнопку **`Button2`** и написать на ней ***«Выход».*** С помощью свойства **`Font`** можно изменить размер шрифта и написание текста на кнопке. С помощью свойства **`ForeColor`** можна изменить цвет шрифта см. рис. 2.1.

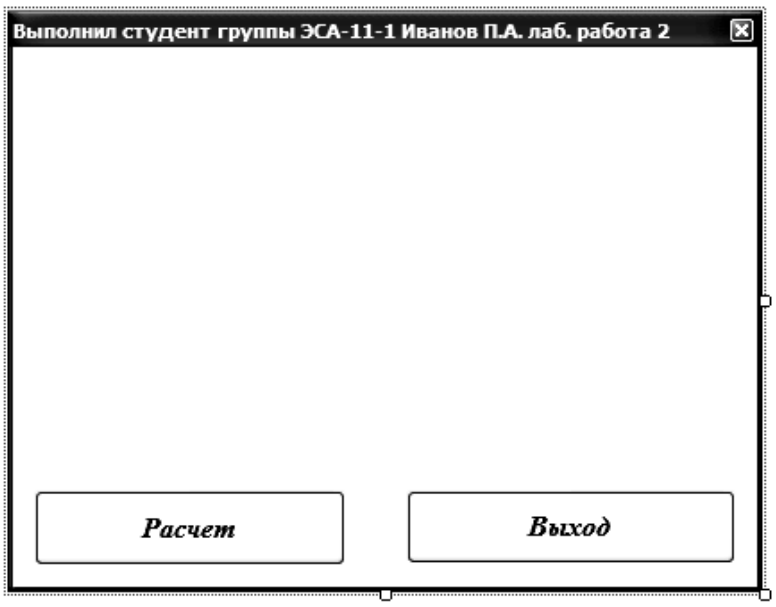


Рисунок 2.1 – Окно формы на этапе создания программы

8 Разместить в самом верху формы компонент **`Label`.** В свойство **`Text`** занести текст «***Вычисление значения функции***» (также можно поменять параметры шрифта).

Для выполнения расчетов необходимо задать исходные данные: значение параметра ***a*** и значение переменной ***x***. Эти данные будут вводиться с клавиатуры с помощью компонента **`TextBox`**. Свойство этого компонента, в которое помещается результат ввода, имеет текстовый тип, т. е. допускает ввод любых символов. Таким образом, в процессе ввода пользователь может ввести (случайно или преднамеренно) недопустимые (нечисловые) символы. Такую ситуацию допустить нельзя, т. к. при вводе недопустимых символов произойдет сбой (аварийное завершение) программы во время преобразования неверных данных к вещественному типу. Поэтому пользователю нужно внимательно вводить данные в эти поля. Для этого выше кнопок разместить три элемента **`TextBox`** (можно один элемент **`TextBox`** вставить, остальные два скопировать с первого). Первые два предназначены для ввода, соответственно, значения параметра **a** и значения переменной **x**. В свойства **`Text`** этих компонентов ввести какие-либо значения – значения по умолчанию. Эти значения будут показываться при запуске приложения на выполнение. При выполнении приложения их можно будет заменить другими. Компонент **`TextBox3`** будет использован для вывода результата вычисления функции, поэтому необходимо запретить ввод в него данных пользователем. Для этого свойству **`ReadOnly`** (только чтение) присвоить значение ***`true`***, запрещающее пользователю заносить в компонент какие-либо данные.

9 Напротив первого элемента **`TextBox1`** разместить элемент **`Label2`** и в свойстве ***`Text`*** этого элемента занести текст «***Введите значение A***», напротив **`TextBox2`** разместить элемент **`Label3`** и в свойстве ***`Text`*** занести текст «***Введите значение X***» и напротив последнего **`TextBox3`** разместить элемент **`Label4`** и в свойстве ***`Text`*** занести текст «***Значение функции У=***». Получим следующую форму на рис. 2.2.

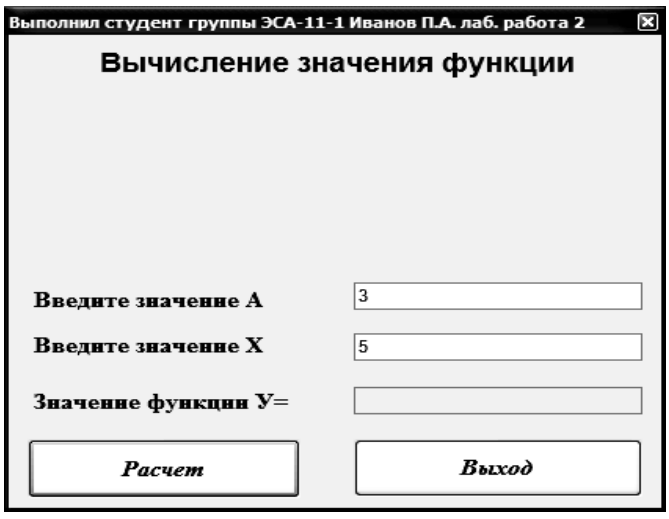


Рисунок 2.2 – Примерный вид проектируемой формы

10 Запустить текстовый процессор **Microsoft Word** и в нем набрать математическую формулу в соответствии с заданием с помощью редактора формул. Выделить набранную формулу и скопировать ее в буфер обмена.

11 Запустить графический редактор **Paint** (*Пуск* → *Программы* → *Стандартные* → *Paint*) и вставить из буфера обмена набранную формулу (рис. 2.3).

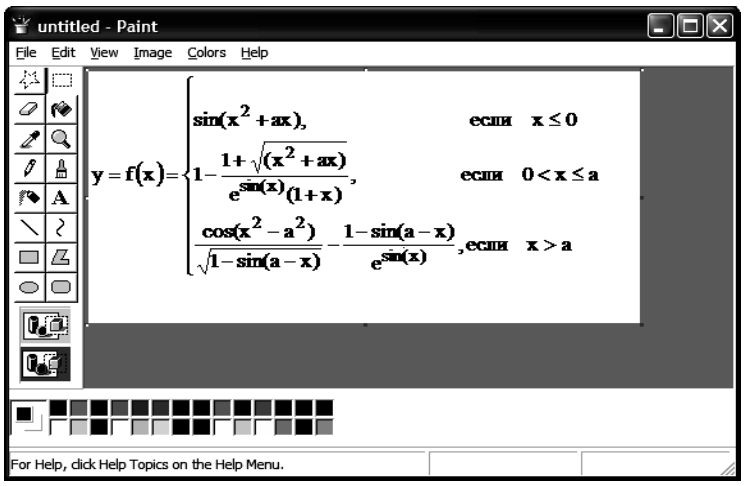


Рисунок 2.3 – Окно графического редактора Paint

12 Сохранить графический файл (на английском языке) в ту же папку, что и проект программы (**Lab2**).

13 Поместить на форму компонент **`PictureBox1`** и с помощью свойства **`Image`** загрузить в него созданный нами рисунок, отображающий функцию. Загрузка рисунка производится с помощью стандартного окна диалога. С помощью свойства **`SizeMode`** вписываем нашу картинку в блок **`PictureBox1`**, выбрав пункт ***`StretchImage`***.

14 В результате проведенных операций должна получиться форма примерно такого вида, как показано на рис. 2.4.

Рисунок 2.4 – Примерный вид формы приложения

15 Создадим событие ***Click*** для кнопки **Button1** с надписью ***«Расчет»***. Для этого сделаем двойной щелчок левой кнопкой мыши по компоненту **Button1**. Получим заготовку подпрограммы:

```
private: System::Void button1_Click(System::Object^  sender,
System::EventArgs^  e) {            }
```

16 Между двумя фигурными скобками вставим следующий фрагмент программного кода:

```
double y, x, a;
//Проверка, что данные для x и a введены
if ((textBox1->Text!="")&&(textBox2->Text!=""))
{
//Преобразование из текста в дробное число x и a
a = Convert::ToDouble(textBox1->Text);
x = Convert::ToDouble(textBox2->Text);
if (x<=0)y=sin(x*x+a*x);
        else
if (x>0&&x<=a){y=1-(1+sqrt(x*x+a*x))/(exp(sin(x))*(1+x));}
        else
{y=(cos(x*x-a*a)/(sqrt(1-sin(a-x)))-(1-sin(a-x))/(exp(sin(x)));};
//Вывод в компоненте TextBox3 преобразованного в текст значения y
textBox3->Text = Convert::ToString (y);}
        else
//Вывод окна с сообщением, если не введены данные в компоненты TextBox1 и TextBox2
{MessageBox::Show( "Введите пожалуйста данные A и X", "Ошибка
ввода   данных",   MessageBoxButtons::OK,          MessageBoxI-
con::Exclamation); }
```

17 Создадим событие ***Click*** для кнопки **Button2** с надписью ***«Выход»***. Для этого сделаем двойной щелчок левой кнопкой мыши по компоненту **Button2**. Получим заготовку подпрограммы:

```
private: System::Void button2_Click(System::Object^  sender,
System::EventArgs^  e) {       }
```

18 Между двумя фигурными скобками вставим следующий фрагмент программного кода:

```
Application ::Exit();
```

19 Вверху программы после слов `#pragma once` добавим строку, подключив библиотеку работы с математическими функциями

```
#include <math.h>,
```

а также добавим строку, подключив библиотеку работы со строковыми функциями:

```
#include <string.h>
```

20 При создании прикладных программ разработчики делают максимально доступным пониманию интерфейс, а саму программу максимально «дружественной» к действиям пользователя. В нашем создаваемом приложении пользователь при вводе исходных данных (числовых значений ***a*** и ***x***) может допустить ошибку и ввести текст вместо числа. Необходимо создать процедуру обработки события ***Потери фокуса*** компонентами **TextBox1** и **TextBox2** (данное событие происходит в момент, когда

курсор переходит от текущего однострочного текстового поля к другому объекту). Выделим на создаваемой форме объект **TextBox1** и перейдем в окно ***Свойства*** на вкладку ***События***  . Найдем интересующее нас событие **Leave** и двойным щелчком левой кнопки мыши справа от названия события (в пустом белом поле) создадим заготовку подпрограммы:

```
private: System::Void textBox1_Leave(System::Object^  sender,
System::EventArgs^  e)
     {
     }
```

21 Между двумя фигурными скобками вставим следующий фрагмент программного кода:

```
int l, t, k; bool a=true; String ^str;
str=textBox1->Text;
l=str->Length;

//индекс символа, с которым работаем
t=0;
//колличество запятых в строке (дабы избежать варианта
0,2384,1254,1251 – это не число)
k=0;
//двигаем индексатор, если наше число отрицательное
if(str[t]=='-') t++;
//число не может начинаться с запятой
if(str[t]==',') a=false;
while(t<l)
     { if(str[t]==',')
//если запятая стоит последним символом или запятая уже была найдена
{ if(t==l-1 || k>0) a=false;
            k++;
          }
//если t-ый символ не лежит в диапазоне от '0' до '9'
else if(str[t]<'0' || str[t]>'9') a=false;
         t++;
       }
if (a==false)
     { MessageBox::Show("параметр A не является числом","Ошибка
ввода данных",MessageBoxButtons: :OK, MessageBoxIcon:
:Exclamation);
//возврат фокуса текстовому полю
      this->textBox1->Focus();
     }
```

22 Аналогичным образом создаем процедуру, проверяющую на выходе соответствие числовой записи данных в поле **TextBox2**. Вновь создаем подпрограмму, обрабатывающую событие **Leave**, с указанным ниже программным кодом:

```
int l, t, k; bool a=true; String ^str;
str=textBox2->Text;
l=str->Length;
//индекс символа,  с которым работаем
```

```
t=0;
```
//колличество запятых в строке (дабы избежать варианта 0,2384,1254,1251 – это не число)
```
k=0;
```
//двигаем индексатор, если наше число отрицательное
```
if(str[t]=='-') t++;
```
//число не может начинаться с запятой
```
if(str[t]==',') a=false;
while(t<l)
     { if(str[t]==',')
```
//если запятая стоит последним символом или запятая уже была найдена
```
        { if(t==l-1 || k>0) a=false;
            k++;
          }
```
//если t-ый символ не лежит в диапазоне от '0' до '9'
```
          else if(str[t]<'0' || str[t]>'9') a=false;
          t++;
        }
if (a==false)
     { MessageBox::Show("параметр X не является числом","Ошибка
ввода данных", MessageBoxButtons::OK, MessageBoxIcon:
:Exclamation);
```
//возврат фокуса текстовому полю
```
      this->textBox2->Focus();
     }
```

23 Еще одна «некорректность» программы возникает в момент, когда пользователь, получив ответ для одного значения X, изменяет значение аргумента, чтобы получить другой ответ. До того как пользователь нажмет на кнопку «***Расчет***», на форме отображаются новое значение аргумента и старое значение функции (что является «ошибкой»). Необходимо в момент установки курсора в текстовые поля **TextBox1** или **TextBox2** очищать от старой записи текстовое поле **TextBox3**. Это выполняется путем создания подпрограммы обработки события ***Получение фокуса*** (**Enter**). Выделим на создаваемой форме объект **TextBox1** и перейдем в окно ***Свойства*** на вкладку ***События***. Найдем интересующее нас событие **Enter** и двойным щелчком левой кнопки мыши справа от названия события (в пустом белом поле) создадим заготовку подпрограммы:

```
private: System::Void textBox1_Enter(System::Object^  sender,
System::EventArgs^  e)
     {
     }
```

24 Между двумя фигурными скобками вставим следующий фрагмент программного кода:

```
textBox3->Text = "";
```

25 Аналогичным образом создаем подпрограмму ***Получение фокуса*** (**Enter**) для однострочного тестового поля **TextBox2**, поместив в нее такую же команду (очистка **TextBox3** от текста).

26 Таким образом получим полноценную программу для вычисления значения функции, используя условный оператор `if`.

27 Запустим программу на выполнение, нажав на функциональную кнопку **F5,** и получим следующий вид окна (рис. 2.5).

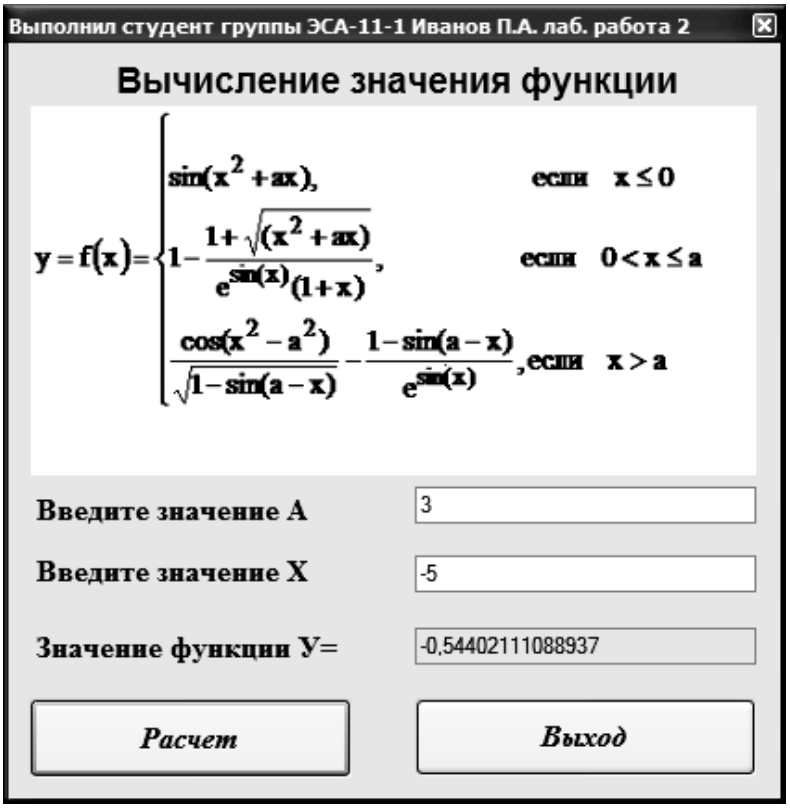


Рисунок 2.5 – Рабочий вид формы приложения

2.3 Рабочее задание

Создать приложение для вычисления и вывода на экран значения функции:

$$y = f(x) = \begin{cases} f_1(x), если & x \le 0 \\ f_2(x), если & 0 < x \le a \\ f_3(x), если & x > a \end{cases}$$

Выражения для функции $f_1(x)$, $f_2(x)$ и $f_3(x)$ выбрать из таблицы 2.4 в соответствии с номером своего варианта. В форме предусмотреть поля для ввода значения параметра **a** и переменной **x**, вывода результата вычисления **y**, а также командные кнопки для осуществления расчета и выхода из приложения.

Таблица 2.4 – Индивидуальные задания

Вариант	$f_1(x)$	$f_2(x)$	$f_3(x)$
1	2	3	4
1	$\lvert x\rvert^{2x+1}$	$\sin x^2$	$\ln^2\lvert x\rvert+\sqrt{x}$
2	$\sin^2 x^3$	$\sqrt[5]{6x-x^2+1}$	$2\sin(x-e^{-x})$
3	$2xe^{-x}$	$(x-1)^3+\cos(x^3)$	$2\sqrt{x^3}\sin(x^3)$
4	$\ln(x^2+5)$	$\sin(e^x+2)$	$\operatorname{tg}(5x+1)$
5	$2\sqrt{\lvert x^3\rvert}\sin(x^3)$	$(x+1)^2\cos x^3$	$\sqrt{x^4+2}+\sin x^2$
6	$\cos x+x^3$	$\sqrt{x^3}\sin x$	$8+\cos(3x)$
7	$x\sin(x+4)$	$\ln(4x^2+1)$	$\ln\sqrt[5]{5+x^2}$
8	x^4+2x^3-x	$1{,}3\sqrt{4+x^2}$	$\lvert x+1\rvert^x$
9	$\lvert x\rvert^5\operatorname{ctg}\lvert x\rvert$	$\ln(x^2+1)$	$e^{-2x}-\sqrt[3]{\lvert x+1\rvert}$
10	$x^5\operatorname{ctg}(2x^3)$	$\sqrt[5]{x^4+3}$	$\lvert\sin^2 x+1\rvert^{2x}$
11	$\operatorname{ctg}(3x-1)^2$	$2+xe^{-x}$	$\sin^3 x^2$
12	$x\sin(x-1)$	$(x-1)^3+\cos x^3$	$\sqrt{\lvert x\rvert^3}\sin x^3$
13	$(x+1)/(\lvert x\rvert+2)^3$	$e^x+\cos(x+2)$	$3\ln\sqrt[5]{\sin^2 x+2}$
14	$3x^5-\operatorname{ctg}x^3$	$\ln(\sin 4x+1)^2$	$\sqrt[3]{2x^2+x^4+1}$
15	$1{,}3\sqrt{4+x^2}$	3^{x+3}	$x^{x+1}\sin(x+2)$
16	$e^{-3x}+\cos x$	$\sin^3 x^4$	$e^{-x}+\sqrt[3]{3x^2+1}$
17	$x^3+(\lvert x\rvert+1)^{0,1x}$	$(x-1)^3+\cos(2x^3)$	$\sin(7x)+\operatorname{tg}(0{,}01x)$
18	$\operatorname{tg}(0{,}1\pi x^2)+x$	$e^{x+1}-\sin(x+\pi)$	$3\sqrt[5]{\sin^2 x+2}$
19	$3x^5-\operatorname{ctg}(\pi x^3)$	$(x+1)^{0,3}+\sin 2x^3$	$5x-x^2$
20	$\lvert x\rvert^{\sin(x)}+\sin(x)$	$3^{x+3}+2x$	$2^x+\sin(\pi x)$
21	$x^2+\sin(7x)$	$\lvert x^3+10^x\rvert$	$\sqrt[7]{2x^4+x^2+1}$
22	$\left\lvert\sqrt[3]{\dfrac{2x+5}{(x^3+2)}}\right\rvert$	$\dfrac{5x+x^2}{(x^2+3)^3}$	$\cos^2(x^3+\sqrt{x})$
23	$\sqrt[5]{x^2+x+1}$	$\ln^2(\lvert\sqrt{x+5}\rvert)$	$\sin(x^2)+x^{0,25}$

Продолжение таблицы 2.4

1	2	3	4
24	$\sqrt[3]{\lvert x\rvert + 2} - 1$	$\sin(x^3) + x^{0,5}$	$\ln^2(x) + \sqrt{x}$
25	$\sqrt{\sin^2 x + \cos^4 x}$	$\ln^2(x) + \sqrt{x}$	$\mathrm{tg}^2(x) + \sqrt{x}$
26	$x^3 - \ln(\lvert x\rvert + 1)$	$\frac{2x+2}{(\mathrm{tg}(2x-1)+1)}$	$x^4 - x^x$
27	$x^4 + 2x^3 - x$	$e^{-x} + \sqrt[4]{x}$	$\ln(x^3 + x^2)$
28	$\frac{(3x-1)^2}{x^5}$	$\ln^2\lvert\sqrt{x+5}\rvert$	$\cos\left(\sqrt{1+x^2}\right)$
29	$x^5 \mathrm{ctg}(2x^3)$	$\frac{5}{\mathrm{tg}(2x+3)+1}$	$\mathrm{tg}\left(x^2+1\right)e^{-x}$
30	$\lvert x\rvert\sin(3x)$	$x^3\cos(x+2)$	$\sin x^2 + x^{0.25}$

2.4 Контрольные вопросы

1 Что такое полная и сокращенная форма условного оператора **`if`**?

2 Как организуются множественные действия в операторе условия **`if`**?

3 Какой формат записи имеет тернарный оператор условия?

4 В чем различие и сходство между операторами break и continue?

5 Как можно обеспечить выход из вложенных циклов?

6 Как можно организовать переходы в различные точки программы на **C++**?

7 Какие логические операторы отношения используются в языке **C++**?

8 Как сделать, чтобы на форме отображалась картинка математической функции?

9 Как записать математические функции на языке программирования C++?

10 Как осуществляется проверка ввода данных для ***x*** и ***a***?

3 ЛАБОРАТОРНАЯ РАБОТА 3. ЦИКЛИЧЕСКИЙ АЛГОРИТМ. ТАБУЛИРОВАНИЕ ФУНКЦИИ И ПОИСК ЭКСТРЕМУМОВ

Цель: изучение операторов цикла; создание циклических алгоритмов; приобретение навыков создания проекта вывода данных в виде таблицы в ***Visual Studio 2010***.

3.1 Теоретические сведения

Цикл – разновидность управляющей конструкции высокоуровневых языков программирования, предназначенная для организации многократного исполнения набора инструкций. Также циклом может называться любая многократно исполняемая последовательность инструкций, организованная любым способом (например, с помощью условного перехода).

Последовательность инструкций, предназначенная для многократного исполнения, называется *телом цикла*. Единичное выполнение тела цикла называется итерацией. Выражение определяющее, будет в очередной раз выполняться итерация или цикл завершится, называется *условием выхода* или *условием окончания цикла* (либо *условием продолжения* в зависимости от того, как интерпретируется его истинность: как признак необходимости завершения или продолжения цикла). Переменная, хранящая текущий номер итерации, называется *счётчиком итераций* цикла или просто *счётчиком цикла*. Цикл не обязательно содержит счётчик, счётчик не обязан быть один – условие выхода из цикла может зависеть от нескольких изменяемых в цикле переменных, а может определяться внешними условиями (например, наступлением определённого времени), в последнем случае счётчик может вообще не понадобиться.

Исполнение любого цикла включает первоначальную инициализацию переменных цикла, проверку условия выхода, исполнение тела цикла и обновление переменной цикла на каждой итерации. Кроме того, большинство языков программирования предоставляют средства для досрочного управления циклом, например, операторы завершения цикла, то есть выхода из цикла независимо от истинности условия выхода (в языке Си – `break`) и операторы пропуска итерации (в языке Си – `continue`).

3.1.1 Оператор `while`

Изучение операторов цикла начнем с оператора `while`. Цикл `while` имеет следующий формат (синтаксис) записи:

```
while (условие)
действие;
```

Производится расчет выражения условие, заключенного в круглые скобки. Если в результате расчета выражения условие получается истинный результат (TRUE), то выполняется утверждение действие, следующее непосредственно за закрывающей круглой скобкой. После выполнения этого утверждения вновь рассчитывается выражение условие. Если в результате расчета будет TRUE, то вновь будут выполнены утверждения действие. Цикл повторяется до тех пор, пока в результате расчета выражения условие (в круглых скобках оператора while) не будет получено значение FALSE (ложный), которое является признаком окончания цикла, после чего выполнение программы продолжается с утверждения, следующего за утверждением действие. Когда требуется выполнить группу утверждений, то она (группа) располагается в фигурных скобках:

```
while (условие)
{
действие;
действие2;
действие3;
...
}
```

Открывающаяся фигурная скобка может следовать непосредственно после закрывающей круглой скобки оператора while. Все, что находится в фигурных скобках, будет выполняться, пока верно выражение условие.

Очевидно, что неверное задание выражения условие может привести к бесконечному циклу (к зацикливанию).

3.1.2 Оператор `for`

Оператор цикла for имеет следующий формат записи:

```
  for (нач_знач; условие; счетчик)
действие;
```

Три выражения, заключенные в круглые скобки оператора цикла for, задают условия выполнения программного цикла.

Первый параметр нач_знач используется для задания начального значения цикла.

Второй компонент условие определяет условие или условия, в соответствии с которыми будет происходить выход из цикла. Повторение будет происходить до тех пор, пока это условие (или условия) выполняется. Если условие не выполняется, то цикл немедленно заканчивается.

Третий параметр счетчик выполняется каждый раз, когда заканчивается обработка тела цикла, т. е. действие.

Чаще всего выражения init_expression и loop_expression являются операторами присваивания или вызовами функций, а второе выражение

loop_condition – выражением отношения или логическим выражением. Любую из трех частей можно опустить, но точки с запятыми должны остаться на своих местах. Если опустить init_expression или loop_expression, то соответствующие операции не будут выполняться. Если же опустить проверку условия loop_condition, то по умолчанию считается, что условие продолжения цикла всегда истинно, и тогда цикл станет бесконечным (произойдет зацикливание).

Когда требуется выполнения нескольких утверждений, то они должны заключаться в фигурные скобки:

```
for (init_expression; loop_condition; loop_expression)
{
действие1;
действие2;
действие3;
...
}
```

В представленном случае тело цикла находится в фигурных скобках.

Конструкция цикла, реализованная оператором `for`, может быть выполнена также и оператором `while` следующим образом:

```
init_expression;
while (loop_condition)
{
program statement;
loop_expression;
}
```

Исключением является применение операции continue.

В программах языка **C** возможно применять вложенные циклы, каждый из которых контролируется своей переменной цикла и своим отношением (второе выражение в круглых скобках оператора for). Вложенные циклы могут идти непосредственно друг за другом или составлять тело цикла с помощью фигурных скобок. Возможно также использование двух индексных переменных для инициализации начала цикла с последующим их инкрементированием (увеличением) или декрементированием (уменьшением).

3.1.3 Оператор `do-while`

Рассмотренные операторы цикла while и for производят проверку условия выполнения цикла до начала выполнения тела цикла. Поэтому тело цикла может ни разу не выполниться, если с самого начала результатом расчета условия выполнения цикла будет значение FALSE (ложь). В случае необходимости производить проверку условия выполнения цикла после тела цикла (т. е. когда выполняется хотя бы одно предписанное действие в теле цикла) прибегают к циклу do–while.

Оператор цикла do–while имеет следующий формат записи:

```
do
program statement;
while (loop_expression);
```

Выполнение цикла do–while происходит следующим образом: сначала выполняется утверждение program statement, затем производится проверка условия выполнения цикла loop_expression с помощью оператора while. Если результатом проверки будет значение TRUE (истина), то выполнение цикла продолжится, и утверждение program statement всякий раз будет выполняться вновь. Повторение цикла будет продолжаться до тех пор, пока в результате проверки условия выполнения цикла loop_expression будет получаться значение TRUE. Когда в результате проверки условия будет вычислено значение FALSE (ложь), то выполнение цикла прекратится и произойдет переход к утверждению (следующему фрагменту программы), непосредственно следующему за циклом.

Таким образом, цикл do–while гарантированно выполнится хотя бы один раз.

В случае выполнения нескольких утверждений используются фигурные скобки для выделения тела цикла:

```
do {
program1 statement1;
program2 statement2;
program3 statement3;
... } while (loop_expression);
```

Оператор цикла while называется оператором цикла с предусловием, оператор цикла for называется оператором цикла с параметром, оператор цикла do–while называется оператором цикла с постусловием.

Алгоритм циклической структуры – это вычислительный процесс, содержащий многократные вычисления по одним и тем же математическим зависимостям, но для различных значений входящих в него переменных. Переменные, изменяющиеся в цикле, называются параметрами цикла.

Каждый алгоритм циклической структуры содержит такие элементы:

а) ***подготовка цикла*** – определение начальных значений параметров цикла;

б) ***тело цикла*** – действия, что повторяются многократно для разных значений параметров цикла;

в) ***модификация цикла*** – смена значений параметров цикла;

г) ***управление циклом*** – проверка условия выхода из цикла.

Табулирование функции – это формирование и вывод на экран или принтер таблицы значений функции для значений аргумента (**x**), изменяющихся от некоторого начального значения ($\mathbf{x_n}$) до конечного ($\mathbf{x_к}$) с некоторым шагом (**h**). Для этого используется цикл. При подготовке цикла аргументу присваивается начальное значение ($\mathbf{x = x_n}$), в теле цикла вычисляются и выводятся значения функции для текущего значения аргумента **x**. Модификация заключается в увеличении аргумента на величину **h** (**x = x + h**). Цикл завершается, когда после очередного изменения значение аргумента превысит конечное значение ($\mathbf{x > x_к}$).

Экстремум – это наибольшее или наименьшее значение функции на промежутке.

Поиск экстремумов функции на заданном отрезке методом перебора выполняется в цикле, аналогично табулированию. Вместо вывода значение функции в каждой точке сравнивается с наибольшим (наименьшим) из значений во всех предыдущих точках. Если текущее значение больше (меньше) наибольшего (наименьшего) из предыдущих, то его надо считать новым наибольшим (наименьшим) значением. В противном случае наибольшим (наименьшим) значением остается значение функции, определенное в предыдущих точках.

3.2 Пример выполнения работы

Создать Windows-приложение для вычисления значения функции и нахождение экстремумов функции

$$y = f(x) = \begin{cases} 2x + 2, & \textit{если} \quad x \le 0 \\ \sqrt{x+3}, & \textit{если} \quad 0 < x \le a \\ \cos^2(x+2), & \textit{если} \quad x > a \end{cases}.$$

с использованием оператора `While` на отезке **[xn;xk]** с шагом **xh.**

1 Войти в среду ***Visual Studio 2010*.**

2 В окне **Создать Проект** следует развернуть узел `Visual C++,` обратиться к пункту `CLR` и на центральной панели выбрать *Приложение* `Windows Form`.

3 Затем в поле редактора **Имя** (где по умолчанию имеется <Введите имя>) следует ввести имя проекта, **Visual_Lab3**. В поле **Расположение** можно указать путь размещения проекта, или выбрать путь размещения проекта с помощью клавиши (кнопки) **Обзор** (например, N:\\CI\2_trim\Lab3).

4 Для формы изменить значение свойства **Text,** занеся, например, следующие данные: ***«Выполнил студент группы ЭСА-11-1 Иванов П. А. Лабораторная работа 3».***

5 Свойству **`FormBorderStyle`** формы (стиль рамки окна) присвоить значение ***`FixedToolWindow`***. Это значение определяет окно как диалоговое, его размеры на этапе прогона приложения (в процессе работы приложения) не могут быть изменены.

6 Поместить на форму два компонента **`Button1`** и **`Button2`** (кнопка), выше этих компонентов левее центра разместить четыре поля для ввода данных: **`TextBox1`**, **`TextBox2`**, … , **`TextBox4`**.

7 Напротив компонентов **TextBox** разместить четыре элемента: **Label1**, **Label2**, …, **Label4**.

8 В верхнем левом углу формы расположить компонент **PictureBox1** (блок для картинки).

9 Напротив компонента **PictureBox1** разместить компонент **DataGridView1** (таблица для просмотра данных).

10 Над компонентом **PictureBox1** разместить элемент **Label5**.

11 Под компонентом **PictureBox1** разместить два компонент **Label6** и **Label67**, а под ними два компонента для вывода экстремумов функции **TextBox5**, **TextBox6**. Получим следующую форму с размещенными на ней компонентами (рис. 3.1).

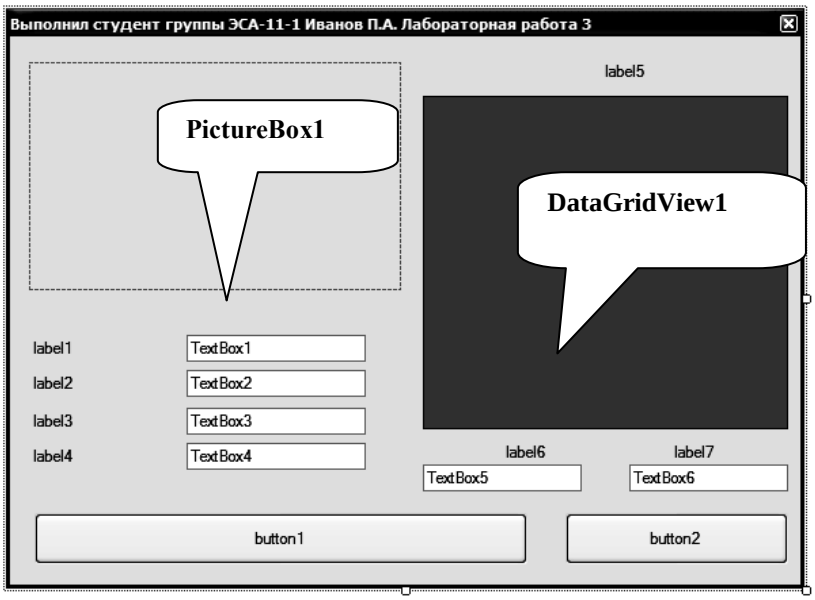


Рисунок 3.1 – Окно формы на этапе создания программы

11 Выделить командную кнопку **Button1** и написать на ней ***«Показать таблицу и экстремумы функции»*** (т. е. занести этот текст в свойство **Text**), аналогично для кнопки **Button2** сделать надпись ***«Выход».***

12 Выделить компонент **Label1** и в свойство **Text** занести текст «***Введите значение XN=***» (начальное значение интервала), аналогично для **Label2** в свойство **Text** занести текст «***Введите значение XK=***» (конечное значение интервала), для **Label3** занести текст «***Введите значение XH=***» (шаг изменения значения **X**), для **Label4** в свойство **Text** занести текст «***Введите значение a=***» (положительное число меньшее ***XK***).

13 Выделить компонент **Label5** и в свойство **Text** занести текст «***Таблица значений функции***».

14 Выделить компонент **Label6** и в свойство **Text** занести текст «***Максим. знач. функции***», аналогично для **Label7** в свойство **Text** занести текст «***Миним. знач. функции***».

15 Для выполнения расчетов необходимо задать исходные данные: значение параметра **a** и значения переменных **XN, XK, XH**. Эти данные будут вводиться с клавиатуры с помощью компонента однострочный редактор класса **`TextBox`**. В свойства **`Text`** этих компонентов ввести какие-либо значения – значения по умолчанию. Эти значения будут показываться при запуске приложения на выполнение. При выполнении приложения их можно будет заменить другими.

16 Компоненты **`TextBox5`** и **`TextBox6`** будут использованы для вывода результатов нахождения экстремумов функции (максимального и минимального значения функции). Поэтому необходимо запретить ввод в него данных пользователем. Для этого свойству **`ReadOnly`** (только чтение) присвоить значение ***`true`***, запрещающее пользователю заносить в компонент какие-либо данные. Получим следующую форму на рис. 3.2.

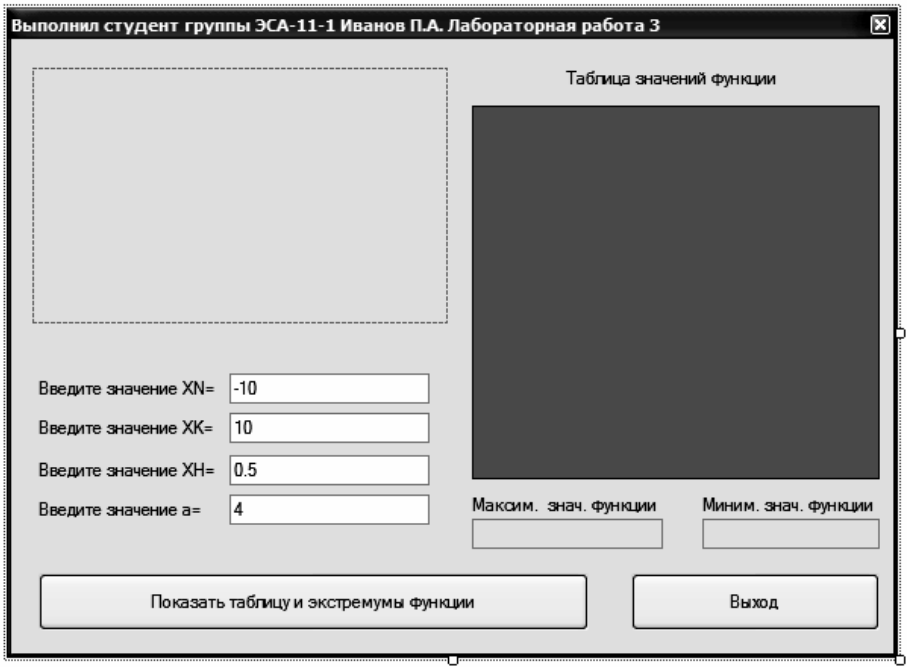


Рисунок 3.2 – Примерный вид проектируемой формы

17 Запустить текстовый процессор **Microsoft Word** и в нем набрать математическую формулу в соответствии с заданием с помощью редактора формул. Выделить набранную формулу и скопировать ее в буфер обмена.

18 Запустить графический редактор **Paint** (*Пуск → Программы → Стандартные → Paint*) и вставить из буфера обмена набранную формулу. Сохранить графический файл (на английском языке) в туже папку, что и проект программы (**Visual_Lab3**).

19 Выделить на форме компонент **`PictureBox1`** и с помощью свойства **`Image`** загрузить в него созданный нами рисунок, отображающий функцию. Загрузка рисунка производится с помощью стандартного окна диалога. С помощью свойства **`SizeMode`** вписываем нашу картинку в блок **`PictureBox1`**, выбрав пункт ***`StretchImage`***.

20 Выделить компонент **DataGridView1** и для свойств **RowHeaders Visible** (отображение заголовка строк) выбрать параметр ***False***.

21 В результате проведенных операций должна получиться форма примерно такого вида, как показано на рис. 3.3.

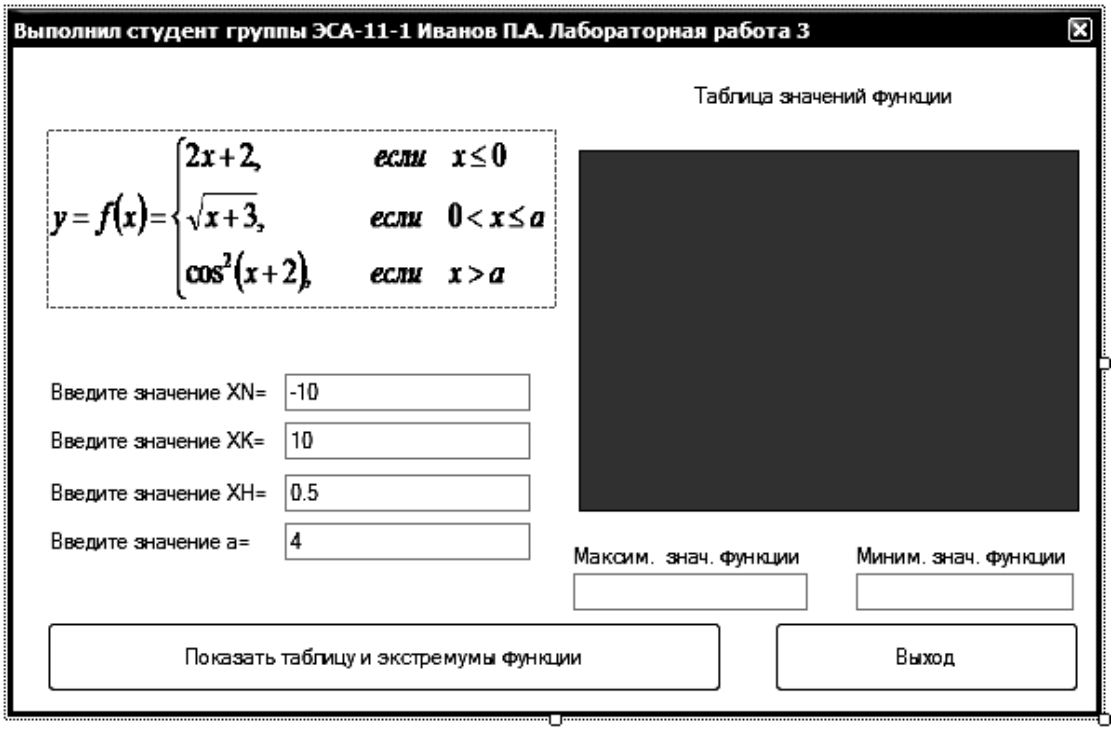


Рисунок 3.3 – Примерный вид формы приложения

22 Перейти к коду программы и в самом верху второй строкой кода программы подключить библиотеку использования математических функций, для этого вставить

```
#include <math.h>
```

23 Создать событие ***Click*** для кнопки **Button1** с надписью ***«Показать таблицу и экстремумы функции»***. Для этого сделать двойной щелчок левой кнопкой мыши по компоненту **Button1**. Внести изменения в код подпрограммы и получить следующий вид программы:

```
private: System::Void button1_Click(System::Object^  sender,
System::EventArgs^  e)
{
     double xn,xk,xh,x,y,a,ymax,ymin,yt;
     int n,i;
//Проверка ввода данных в компоненты textBox

if ((textBox1->Text!="")&&(textBox2->Text!="")&&
(textBox3->Text!="")&&(textBox4->Text!=""))
{
//Преобразование введенных данных в тип double
     xn = Convert::ToDouble(textBox1->Text);
     xk = Convert::ToDouble(textBox2->Text);
     xh = Convert::ToDouble(textBox3->Text);
```

```
     a = Convert::ToDouble(textBox4->Text);
```
//Очистка столбцов таблицы
```
      dataGridView1->Columns->Clear();
```
//Создание двух столбцов в таблице
```
      dataGridView1->ColumnCount = 2;
```
//Создание в таблице строк
```
dataGridView1->Rows->Add(ceil((xk-xn)/xh)+1);
```
//Занесение в верхнюю строку таблицы в первую ячейку текст «X», во вторую текст «У»
```
dataGridView1->Columns[0]->Name="      X";
dataGridView1->Columns[1]->Name="      Y";
i=0;
x=xn;
ymax=-1.8e307;ymin=1.8e307;
while (x<=xk)
          {
     if (x<=0){ y=2*x+2;}
          else
     if (x<=a) {y=sqrt(x+3);}
          else
     {y=pow(cos(x+2),2);}
```
//Занесение в первый столбец значений аргумента X
```
dataGridView1->Rows[i]->Cells[0]->Value =Convert::ToString(x);
```
//Переменной yt присваивает округленное до двух знаков после запятой значение y
```
     yt=ceil(y*100)/100;
```
//Вывод во втором столбце таблицы значение функции У
```
dataGridView1->Rows[i]->Cells[1]->Value
=Convert::ToString(yt);
```
//находит максимальное и минимальное значение и округляет до двух знаков после запятой
```
     if (y>ymax) ymax=ceil(y*100)/100;
     if (y<ymin) ymin=ceil(y*100)/100;
     x=x+xh;
     i++;}
```
//выводит в компоненты textbox максимальное и минимальное значение функции
```
   textBox5->Text = Convert::ToString (ymax);
   textBox6->Text = Convert::ToString (ymin);
}
else {MessageBox::Show( "Заполните, пожалуйста, данные", "Ошибка
ввода данных",
          MessageBoxButtons::OK, MessageBoxIcon::Exclamation
);}
}
```

24 Создать событие ***Click*** для кнопки **Button2** с надписью ***«Выход»***. Для этого сделать двойной щелчок левой кнопкой мыши по компоненту **Button2** и вставить код для выхода из программы (уже использовался в лаб. раб. 2).

25 На примере лабораторной работы 2 сделать проверку на корректность ввода данных **XN**, **XK**, **XH**, **a**, используя событие **`Leave`**, в окне свойств (в код программы добавить 4 процедуры, проверяющие правильность ввода).

26 Запустим программу на выполнение, нажав на функциональную кнопку **F5,** и получим следующий вид окна (рис. 3.4).

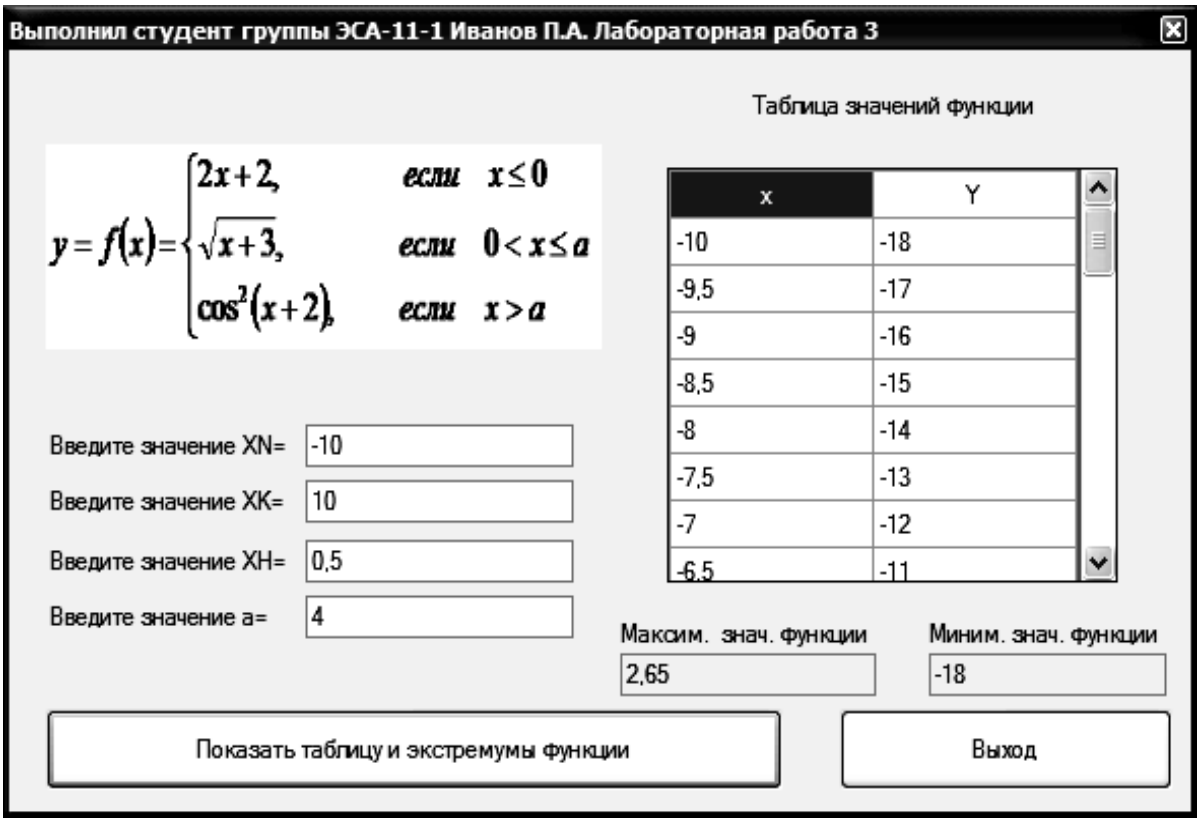


Рисунок 3.4 – Рабочий вид формы приложения

3.3 Рабочее задание

Создать Windows-приложение для вычисления значения функции и нахождение экстремумов функции

$$y=\begin{cases}f1(x), & \text{если } x\le 0\\ f2(x), & \text{если } 0<x\le a\\ f3(x), & \text{если } x>a\end{cases}$$

с использованием оператора `While` на отезке **[xn;xk]** с шагом **xh.** Данные взять из таблицы 3.1.

Таблица 3.1 – Индивидуальные задания

Вариант	Функции			Границы отрезка	Шаг табулирования
	$f1(x)$	$f2(x)$	$f3(x)$		
1	2	3	4	5	6
1	$\sqrt[5]{x^2+x+1}$	$\ln^2(\lvert\sqrt{x+5}\rvert)$	$\sin(x^2)+x^{0,25}$	[–3,9; 3,8]	0,15
2	$3x^5+\operatorname{ctg}(x^3+1)$	$e^{x+1}-\sin(\pi x)$	$\sqrt[5]{\sin^2 x+2}$	[–1,3; 7,1]	0,6
3	$x^5-\operatorname{ctg}(\pi x^3)$	$(\lvert 7x\rvert+1)^{0,3}+\sin x$	$5x-x^2$	[–2,9; 6,2]	0,8
4	$\lvert x\rvert^{x+2}+\sin(x)$	$3^{x+3}+2x$	$\sqrt[5]{x^2+x+1}$	[–3,7; 8,5]	0,11
5	$x^2+\sin(7x)-1$	$\lvert x^3+10^x\rvert$	$\sqrt[7]{2x^4+x^2+1}$	[–3,9; 1,2]	0,25
6	$\sqrt[3]{\lvert x\rvert+2}-1$	$\sin(x^2)+x^{0,25}$	$\ln^2(x)+\sqrt{x}$	[–4,5; 6,1]	0,3
7	$\sqrt{\lvert\sin^2 x+\cos^4 x\rvert}$	$\ln(x+1)+\sqrt{3x}$	$5^{x+1}+\operatorname{tg}(x+3)^2$	[–3,4; 3,4]	0,33
8	$x^3-3x^2\sqrt{\lvert x\rvert+6}$	$\frac{2x+2}{\operatorname{tg}(2x-1)+1}$	x^4-x^x	[–4,1; 5,0]	0,45
9	x^4+2x^3-x	$e^{-x}+\sqrt[3]{x}$	$\ln(\lvert x^3+x^2\rvert)$	[–1,7; 2,9]	0,75
10	$\frac{(3x-1)^2}{x^5+2x+1}$	$\ln^2\lvert\sqrt{x+5}\rvert$	$\sqrt[5]{1+x^2}$	[–1,6; 4,7]	0,65
11	$x^5\operatorname{ctg}(2x^3)$	$\frac{5}{\operatorname{tg}(2x+3)+1}$	x^2e^{-x}	[–1,6;3,7]	0,3
12	$x^x\operatorname{tg}(x+5)$	$x^3\cos x$	$\sin x^2+x^{0.25}$	[–2,8; 8,2]	0,4
13	$\ln\lvert x+1\rvert+\sqrt{3\lvert x\rvert}$	$\sin(x^2+3x)$	$\ln^2\lvert x\rvert+\sqrt{x+3}$	[–1,7; 2,6]	0,25
14	$\sin^2 x^3$	$\sqrt[5]{6x-x^2+1}$	$\sin(x-e^{-x})$	[–2,2; 7,4]	0,23
15	$\cos(x^3+1)e^{-x}$	$\lvert\sqrt[3]{\frac{2x+5}{(x^3+2)}}\rvert$	$2\sqrt{x^2+7\sin(x^3)}$	[–1,1; 7,9]	0,8
16	$2\sqrt{\lvert x^3\rvert}\sin(x^3)$	$(x+1)^2\cos x^3$	$\sqrt{x^4+2}+\sin x^2$	[–1,2; 2,6]	0,1

Продолжение таблица 3.1

1	2	3	4	5	6
17	$\sin\left(x^5+3\right)$	$\sqrt{x^3}\sin\ x$	$x^4-\sin\left(x+1\right)$	[–1,7; 2,4]	0,3
18	$x^4\mathrm{tg}\left(x+2\right)$	$\ln\left(4x^2+1\right)$	$\ln\sqrt[5]{5+x^2}$	[–4,3; 8,0]	0,5
19	$x^5+\sqrt[3]{x+10}$	$1{,}3\sqrt{4+x^2}$	$\left\|x+1\right\|^x$	[–9,1; 5,8]	0,14
20	$\left\|x\right\|^5\mathrm{ctg}\left\|2x\right\|$	$\ln\left(x^2+1\right)$	$e^{-2x}-\sqrt[3]{\left\|x+1\right\|}$	[–3,4; 2,5]	0,23
21	$x^5\mathrm{ctg}\left(2x^3\right)$	$\sqrt[5]{x^4+3}$	$\left\|\sin^2\ x+1\right\|^{2x}$	[–2,2; 8,1]	0,15
22	$\mathrm{ctg}\left(3x-1\right)^2$	$2+xe^{-x}$	$\sin\left(x^3+1\right)$	[–2,8; 5,2]	0,5
23	$x^3+4x^2\sqrt{\left\|x\right\|}$	$(x-1)^3+\cos\ x^3$	$\sqrt{\left\|x\right\|^3}\sin\ x^3$	[–3,2; 7,8]	0,36
24	$(2x+1)(\left\|x\right\|+2)^3$	$e^x+\sin\left(x+2\right)$	$3\ln\ \sqrt[5]{\sin^2 x+2}$	[–6,1; 1,3]	0,15
25	$\mathrm{ctg}\left(x^3+1\right)$	$\ln\left(\sin x+1\right)^2$	$\sqrt[3]{2x^2+x^4+1}$	[–7,4; 0,6]	0,16
26	$1{,}3\sqrt{4+x^2}$	3^{x+3}	$5^{x+1}+\mathrm{tg}\left(x+1\right)$	[–1,2; 7,1]	0,45
27	$e^{2x}+\sin\left(2x^3\right)$	$\sin^3 x^4$	$e^{-x}+\sqrt[3]{3x^2+1}$	[–2,2; 3,9]	0,55
28	$x^3+(\left\|x\right\|+1)^{0,1x}$	$\left(x-1\right)^3+\cos\left(x^3\right)$	$2x+\mathrm{tg}\left(x^2+2\right)$	[–0,3; 4,5]	0,62
29	$\left\|\sqrt[3]{\dfrac{2x+5}{\left(x^3+2\right)}}\right\|$	$\dfrac{5x+x^2}{\left(x^2+3\right)^3}$	$\cos^2\left(x^3+\sqrt{x}\right)$	[–2,4; 4,4]	0,4
30	$\ln\left(x^2+5\right)$	$\sin\ \left(e^x+2\right)$	$\dfrac{\sin\left(x+3\right)}{e^{2x}+\cos\left(x+1\right)}$	[–2,9; 8,2]	0,2

3.4 Контрольные вопросы

1 Оператор цикла с предварительной проверкой условия в языке **C++**.

2 Оператор цикла с последующей проверкой условия в языке **C++**.

3 Сколько параметров требуется для работы оператора цикла с параметром (`For`)?

4 Как реализуется взаимозаменяемость операторов цикла `while` и `for`?

5 В чем сходство и различие между циклами с предусловием и с постусловием?

6 Как в компоненте `DataGridView` отключить скрытые строки.

7 Как `в DataGridView` вывести округленное число до тысячных.

4 ЛАБОРАТОРНАЯ РАБОТА 4. ПОСТРОЕНИЕ ГРАФИКА ФУНКЦИИ НА ПРОМЕЖУТКЕ С ОПРЕДЕЛЕННЫМ ШАГОМ

Цель: научиться создавать Windows-приложения построения графика функции в среде ***Visual Studio 2010*** на определенном промежутке с указанным шагом, используя условия проверки правильности ввода данных.

4.1 Теоретические сведения

4.1.1 Элемент управления `Chart`

Элемент управления `Chart` позволяет создавать диаграммы для сложного статистического или финансового анализа. Этот элемент поддерживает следующие функциональные возможности:

- ряды данных, области диаграммы, оси, условные обозначения, метки и заголовки;
- привязку данных;
- операции с данными: копирование, разбиение, слияние, выравнивание, группирование, сортировку, поиск, фильтрацию и т. д.;
- статистические и финансовые формулы;
- расширенную настройку внешнего вида диаграммы: трехмерную графику, сглаживание, освещение и перспективу;
- события и индивидуальную настройку.

С помощью компонента `Chart` (чертеж) удобнее всего выполнять построение графиков функции по уравнению на промежутке.

Построение графика функции:

1) нужно добавить компонент `Chart` на форму;

2) щелкнуть левой кнопкой по пункту «`Series`» окна «*Свойства*» (рис. 4.1);

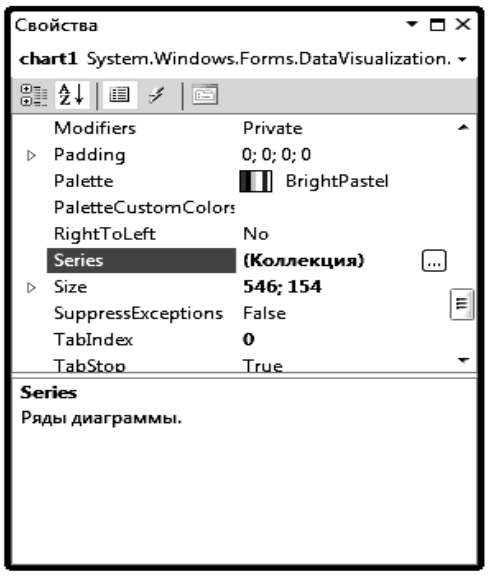


Рисунок 4.1 – Окно «Свойства»

4) в результате увидим окно редактора коллекции `Series` (рис. 4.2);

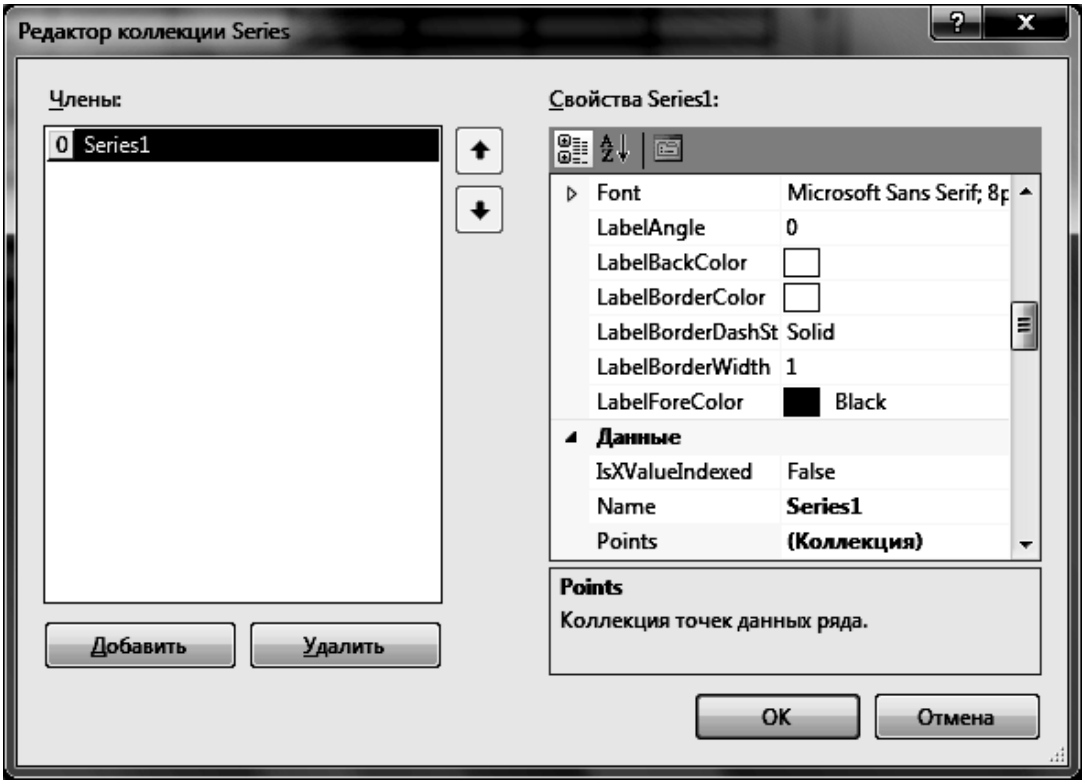


Рисунок 4.2 – Диалоговое окно «Редактор коллекции Series»

5) чтобы на одном компоненте `Chart` построить несколько графиков функции, нужно выбрать страницу `Series` (левая часть окна) и щелкнуть по кнопке *Добавить* (если нужно построить несколько графиков на одной координатной плоскости, то нужно сделать соответствующее количество **`Series`**);

6) в правой части окна для каждой `Series` можно сделать индивидуальные настройки, основные из них:

- `ChartType` – тип диаграммы для представления данных;
- `XValueType` – тип значений, хранимых на оси OX;
- `YValueType` – тип значений, хранимых на оси OY;
- `Color` – цвет точки данных;
- `BorderColor` – задает цвет границ;
- `ShadowColor` – задает цвет тени;
- `Font` – шрифт точки данных;
- `LabelBackColor` – задает цвет фона метки;
- `LabelForeColor` – задает цвет метки;
- `IsVisibleInLegend` – включает/выключает отображение легенды;

Многие из этих параметров можно также задавать программно, например:

```
Series^ plot1 = chart1->Series[0];
if(colorDialog1->ShowDialog()==System::Windows::Forms::DialogResult::OK)
plot1->Color = colorDialog1->Color;
```

В данном случае изменяется цвет графика функции при выборе цвета в диалоговом окне «ColorDialog».

Если графики функций нужно распечатать, то на черно-белом принтере линии будут неразличимыми. В этом случае нужно использовать возможности страницы `Series` и из списка опций `BorderdashStyle` выбрать другую прорисовку линии (например, точками) или в опции `BorderWidth` выбрать большую толщину линии (по умолчанию стоит значение 1).

4.1.1.1 Свойства элемента управления `Chart`

Основные свойства элемента управления `Chart` отображены в таблице 4.1.

Таблица 4.1 – Основные свойства `Chart`

Название свойства	Описание свойства
`BackColor`	Задает цвет фона для объекта `Chart`
`BackHatchStyle`	Задает стиль штриховки для элемента управления `Chart`
`BackImage`	Задает фоновое изображение для элемента управления `Chart`
`BackImageTransparentColor`	Задает прозрачный цвет элемента управления `Chart`
`BorderColor`	Задает цвет границы диаграммы
`BorderDashStyle`	Задает стиль границы
`BorderlineColor`	Задает цвет линии границы
`BorderlineDashStyle`	Задает стиль линии границы
`BorderlineWidth`	Задает толщину линии границы
`BorderWidth`	Задает ширину границы диаграммы
`Font`	Получает свойства шрифта для элемента управления
`FontHeight`	Задает высоту шрифта элемента управления
`ForeColor`	Задает цвет текста элемента управления `Chart`
`Height`	Задает высоту элемента управления
`Series`	Возврат объекта класса `SeriesCollection`
`Size`	Задает размер элемента управления `Chart`
`Visible`	Включает / выключает элемент управления `Chart`

4.1.1.2 Методы элемента управления `Chart`. Основные методы элемента управления `Chart` отображены в таблице 4.2.

Таблица 4.2 – Основные методы `Chart`

Название метода	Описание метода
`DataBind`	Осуществляет привязку данных элемента управления `Chart` к источнику данных
`Focus`	Задает фокус ввода элемента управления
`Hide`	Скрывает элемент управления от пользователя
`OnClick`	Вызывает событие `Click`
`OnEnter`	Вызывает событие `Enter`
`OnLeave`	Вызывает событие `Leave`
`ResetFont`	Сбрасывает свойство `Font` в значение по умолчанию
`ResetForeColor`	Сбрасывает свойство `ForeColor` в значение по умолчанию
`SaveImage(String, ChartImageFormat)`	Сохраняет изображение диаграммы в указанный файл
`Show`	Отображает элемент управления для пользователя

4.1.1.3 События элемента управления `Chart`. Основные события элемента управления Chart отображены в таблице 4.3.

Таблица 4.3 – Основные события `Chart`

Название события	Описание события
`BackColorChanged`	Происходит при изменении значения свойства `BackColor`
`Click`	Происходит при щелчке элемента управления
`DoubleClick`	Происходит, когда элемент управления дважды щелкается
`Enter`	Происходит при входе в элемент управления
`FontChanged`	Происходит при изменении значения свойства `Font`
`ForeColorChanged`	Происходит при изменении значения свойства `ForeColor`
`Leave`	Происходит, когда фокус ввода покидает элемент управления
`MouseClick`	Генерируется при щелчке элемента управления мышью
`Move`	Происходит при перемещении элемента управления
`VisibleChanged`	Происходит при изменении значения свойства `Visible`

4.2 Пример выполнения работы

Создать Windows-приложение, которое предлагает пользователю ввести данные начала промежутка (xn), конца промежутка (xk) и шага изменения переменной (xh) для построения на одной координатной плоскости трех графиков $f_1(x)$, $f_2(x)$, $f_3(x)$:

$$y = f(x) = \begin{cases} 2x + 2, & \text{если} \quad x \le 0 \\ \sqrt{x+3}, & \text{если} \quad 0 < x \le 5 \\ \cos^2(x+2), & \text{если} \quad x > 5 \end{cases}$$

(сделать проверку правильности ввода данных).

Ход выполнения

1 Войти в среду ***Visual Studio 2010*.**

2 В окне **Создать Проект** следует развернуть узел `Visual C++`, обратиться к пункту `CLR` и на центральной панели выбрать *Приложение `Windows Form`*.

3 Затем в поле редактора **Имя** (где по умолчанию имеется <Введите имя>) следует ввести имя проекта, **Visual_Lab4**. В поле **Расположение** можно указать путь размещения проекта или выбрать путь размещения проекта с помощью клавиши (кнопки) **Обзор** (например, N:\\CI\2_trim\Lab4).

4 Для формы изменим значение свойства **`Text`,** занеся, например, следующие данные: ***«Выполнил студент группы ЭСА-12-1 Иванов П. А. Лабораторная работа 4».***

5 Свойству **`FormBorderStyle`** формы (стиль рамки окна) присвоить значение ***`FixedToolWindow`***. Это значение определяет окно как диалоговое, его размеры на этапе прогона приложения (в процессе работы приложения) не могут быть изменены.

6 В верхней части окна разместить **`Label1`** и в свойстве **`Text`** занести текст «***Построение графика функции на интервале [xn;xk]***».

7 Ниже под **`Label1`** разместить компонент **`Chart1`**. Выделить его и зайти в окно свойств; выбать свойство **`Series`** и нажать на троеточие. Появится диалоговое окно (рис. 4.3).

8 Нажать на кнопку `Добавить` и у нас появится **`Series1`** в левой части окна (*каждая серия способна строить на компоненте* **`Chart1`** *новый график, поэтому, если нужно построить несколько графиков на одной координатной плоскости, то нужно сделать соответствующее количество* **`Series`**).

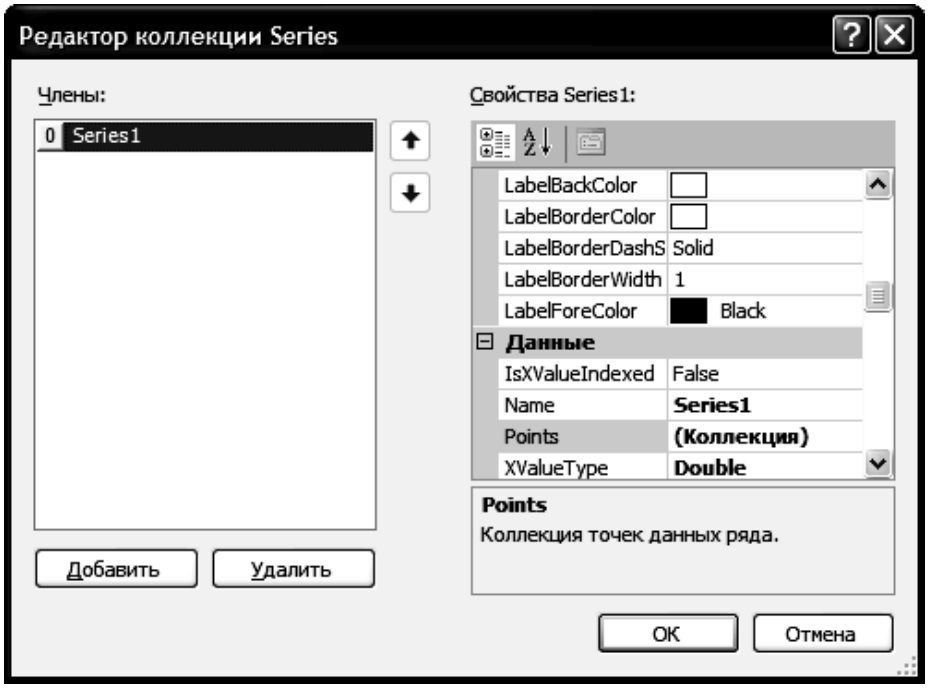


Рисунок 4.3 – Диалоговое окно свойства Series компонента Chart

9 В правой части этого же окна выбрать подпункт «*Данные*» свойства **Series1**, а в нем пункт **XValueType** и с помощью выпадающего списка выбать параметр «***Double***». Аналогично для пункта **YValueType** выбрать параметр «***Double***».

10 Ниже в правой части этого же окна выбрать подпункт «*Диаграмма*» свойства **Series1**, а в нем – пункт **ChartType** и с помощью выпадающего списка выбрать параметр «***Spline***» (*это позволяет соединять точки кривой линией*).

11 И последняя настройка компонента **Chart1**: в правой части этого же диалогового окна выбать подпункт «*Условные обозначения*» свойства **Series1**, а в нем – пункт **IsVisibleInLegend** и с помощью выпадающего списка выбать параметр «***False***» (*это позволяет не отображать легенду в диаграмме*).

12 Ниже компонента **Chart1** разместить четыре компонента **Label**: **Label2**, **Label3**, **Label4**, **Label5**.

13 Выделить **Label2** и в свойстве **Text** занести текст «*Введите интервал построения графика*». Аналогично выделить **Label3** и в свойстве **Text** занести текст «*Введите начальное значение XN=*», для **Label4** – в свойстве **Text** занести текст «*Введите конечное значение XK=*» и для **Label5** в свойстве **Text** занести текст «*Введите значение шага XH=*».

14 Напротив компонента **Label3** разместить компонент **TextBox1**, напротив компонента **Label4** – компонент **TextBox2**, а напротив компонента **Label5** – компонент **TextBox3**.

15 В правом нижнем углу разместить две кнопки **Button1** и **Button2**.

16 Выделить компонент **Button1** и в свойство **Text** занести текст «***Нарисовать график***». При нажатии на эту кнопку будет в компоненте **Chart1** рисоваться график функции $y = \cos(x)$.

17 Выделить компонент **Button2** и в свойство **Text** занести текст «***Выход***».

18 В результате проведенных операций должна получиться форма примерно такого вида, как показано на рис. 4.4.

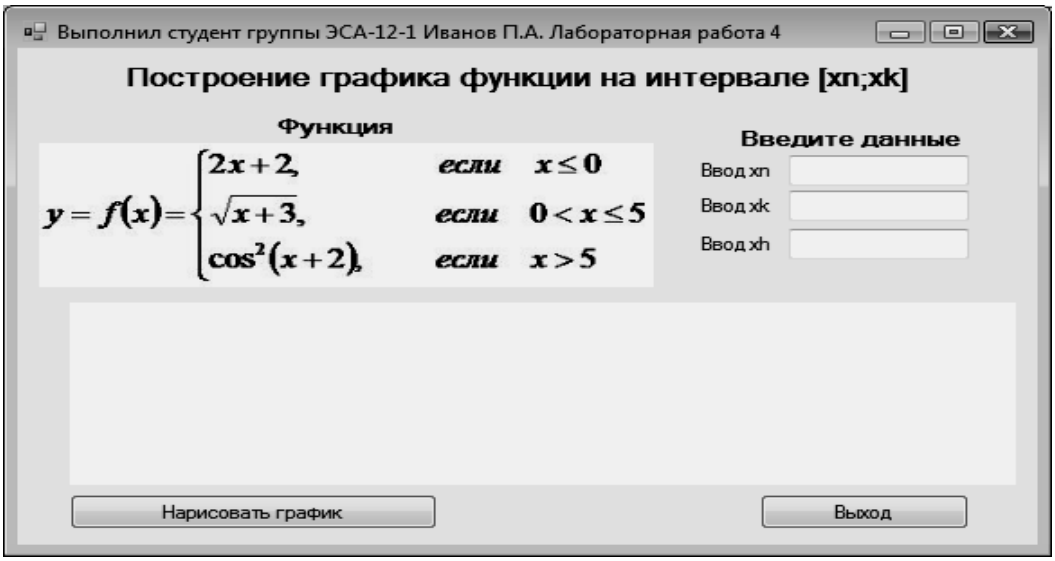


Рисунок 4.4 – Окно формы на этапе создания программы

19 Перейти к коду программы и после строки

```
#pragma once
```

подключить библиотеки использования математических функций, для этого вставить следующую строку:

```
#include<cmath>
```

20 В разделе ***using*** добавить строку:

```
using namespace System::Windows::Forms::DataVisualization::Charting;
```

21 Создадим событие ***Click*** для кнопки **Button1** с надписью ***«Нарисовать график»***. Для этого сделть двойной щелчок левой кнопкой мыши по компоненту **Button1**. Внесем изменения в код подпрограммы:

```
double x,xn,xk,xh,y;
Series^ plot1 = chart1->Series[0];
```

//Очистка компонента Chart1

```
plot1->Points->Clear();
```

//Проверка, что введены данные xn, xk, xh и их преобразование в переменные типа Double

```
if((textBox1->Text!="")&&(textBox2->Text!="")
&&(textBox3->Text!=""))
{
xn = Convert::ToDouble(textBox1->Text);
xk = Convert::ToDouble(textBox2->Text);
xh = Convert::ToDouble(textBox3->Text);
```

//Проверка правильности ввода данных

```
if ((xn>=xk)||(xh>(xk-xn)))
```

```
    {
MessageBox::Show( "Данные заполнены неверно", "Ошибка
ввода данных",MessageBoxButtons::OK,
MessageBoxIcon::Exclamation );
    }
else
     {x=xn;
     while (x<=xk)
{if (x<=0) y=2*x+2;
else
if (x>0&&x<5) y=sqrt(x+3);
else y=pow(cos(x+2),2);
```
//Нанесение точки с координатами X и У в компоненте Chart1
```
plot1->Points->AddXY(x, y);
          x=x+xh;
        }}
}
else
{
MessageBox::Show( "Заполните, пожалуйста, данные", "Ошибка
ввода данных", MessageBoxButtons::OK, MessageBoxIcon:
:Exclamation);
}
```

22 Создадим событие ***Click*** для кнопки **Button2** с надписью ***«Выход»***. Для этого сделать двойной щелчок левой кнопкой мыши по компоненту **Button2** и в полученную заготовку подпрограммы вставить код выхода из программы (уже использовался в предыдущих лабораторных работах).

23 На примере лабораторной работы 2 сделать проверку на корректность ввода данных XN, XK, XH, используя событие **Leave** в окне свойств (в код программы добавить 3 процедуры, проверяющие правильность ввода).

24 Запустить программу на выполнение, нажав на функциональную кнопку **F5**. Получим следующий вид окна (рис. 4.5).

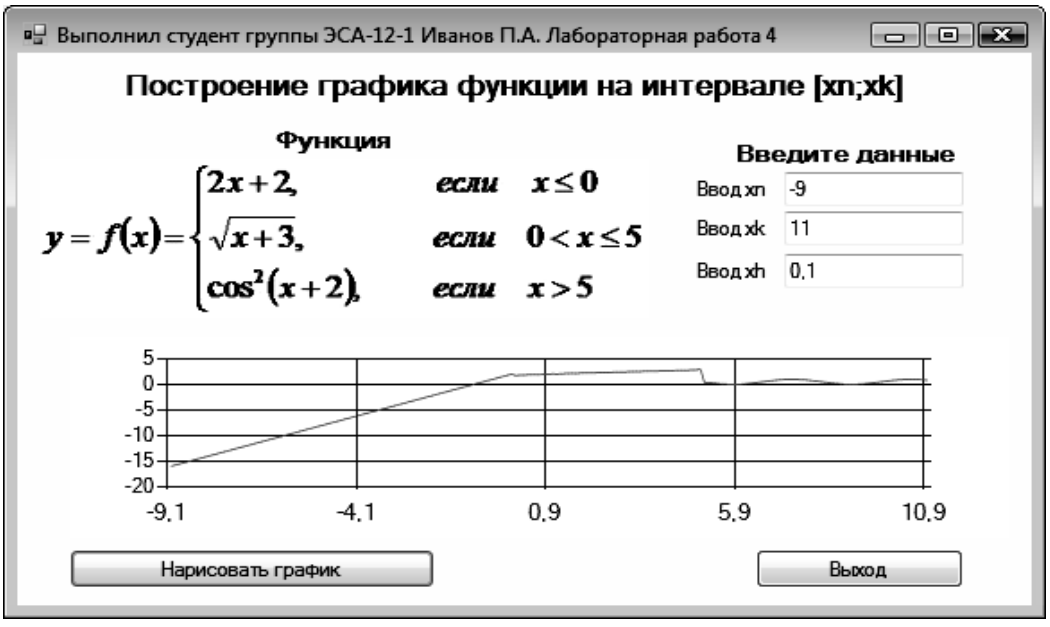


Рисунок 4.5 – Рабочий вид формы приложения

4.3 Рабочее задание

Создать Windows-приложение для построения графиков функций, которое предлагает пользователю ввести данные начала промежутка (xn), конца промежутка (xk) и шага изменения переменной (xh) (сделать проверку правильности ввода данных). На одной координатной плоскости должны быть построены три графика $f_1(x)$, $f_2(x)$, $f_3(x)$ разного цвета.

$$y = f(x) = \begin{cases} f_1(x), если & x \le 0 \\ f_2(x), если & 0 < x \le 5 \\ f_3(x), если & x > 5 \end{cases}$$

Выражения для функций $f_1(x)$, $f_2(x)$ и $f_3(x)$, данные промежутка и шага выбрать из таблицы 4.1 в соответствии с номером своего варианта.

Таблица 4.1 – Индивидуальные задания

Вариант задания	Функции			Начальное значение xn	Конечное значение xk	Шаг (xh)
	$f_1(x)$	$f_2(x)$	$f_3(x)$			
1	2	3	4	5	6	7
1	$\left\|\sqrt[3]{\frac{2x+5}{(x^3+2)}}\right\|$	$\frac{\sqrt{\sin(x^2+3)+4}}{x^2+2}$	$\frac{\sin(x+2)^3}{\ln\left\|x^2+3x+1\right\|}$	2,1	16,5	0,2
2	$\|5\|^{\sin(x)+2}x+\sin(x)$	$x^3+(\|x\|+1)^{0,1x}$	$\frac{\sin^3(x+2)}{\sqrt[4]{\sin^2 x+\cos^4 x}}$	–4,2	28,1	0,1
3	$\frac{\mathrm{tg}(x+3)^2}{\|x\|^{1,2}\sin 3x}$	$\frac{x^3-4x+2}{x^2+\sin(7x)-1}$	$\frac{tg(0,1\pi x^2)+x}{\cos^2(2x+3)}$	-1,7	45,3	0,3
4	$\frac{\cos(x^3-4x+4)}{x^3-\ln(\|x\|+1)}$	$\frac{\sin(x+2)^2}{\sqrt[3]{2x^2+x^4+1}}$	$\frac{\sqrt{\|x\|^3}\sin x^3}{\cos^2(x+1)}$	-2,25	34,9	0,5
5	$\frac{\sin^2(x+5)^2}{e^{-x}+\sqrt[3]{3x^2+1}}$	$\frac{x^4+2x^3-x}{\cos(x+3)^2}$	$\frac{tg(x^2+4x-1)}{2\sqrt{x^3}\sin(x^3)}$	2,45	25,2	0,1
6	$\|x\|^5 ctg(x+2)$	$\frac{5x+x^2}{(x^2+3)^3}$	$\frac{\sin^2(x+3)}{x^5-ctg(\pi x^3)}$	3,35	36,26	0,2

Продолжение таблицы 4.1

1	2	3	4	5	6	7
7	$\frac{\sin(x+3)}{x^5 ctg(2x^3)}$	$\frac{\lvert x\rvert+2}{\cos^2(x^3+2x+1)^2}$	$\frac{\sin^2(x+5)}{\sqrt[3]{\lvert x\rvert+2}-1}$	1,7	4,9	0,5
8	$\frac{e^x \ln\lvert x\rvert}{ctg(3x-1)^2}$	$\frac{5}{tg(2x+3)+1}$	$\frac{(3x-1)^2}{x^5+\sin(x+2)^3}$	–5,2	11,7	0,5
9	$e^{-2x}+\sqrt[7]{2x^4+x^2+1}$	$\frac{x^3+2x^2-4x}{x^5 ctg(2x^3)}$	$\frac{\cos^2(x+2)^3}{2\sqrt{x^3}\sin(x^3)}$	2,9	17,48	0,3
10	$\frac{tg^2(x+1)}{x^4+2x^3-x}$	$\frac{2x+2}{tg(2x-1)+1}$	$\frac{\cos(x+2)^2}{e^{-2x}+\sqrt[4]{3x^2+1}}$	–1,9	29,7	0,1
11	$\sqrt{\sin^2 x+\cos^4 x}$	$\ln^2(x)+\sqrt{x}$	$\mathrm{tg}^2(x)+\sqrt{x}$	–2,74	28,29	0,1
12	$x^3-\ln(\lvert x\rvert+1)$	$\frac{2x+2}{(\mathrm{tg}(2x-1)+1)}$	x^4-x^x	–1,25	9,39	0,4
13	x^4+2x^3-x	$e^{-x}+\sqrt[4]{x}$	$\ln(x^3+x^2)$	–1,78	11,99	0,5
14	$\frac{(3x-1)^2}{x^5}$	$\ln^2\lvert\sqrt{x+5}\rvert$	$\cos\left(\sqrt{1+x^2}\right)$	–2,46	28,8	0,6
15	$x^5\mathrm{ctg}(2x^3)$	$\frac{5}{\mathrm{tg}(2x+3)+1}$	$\mathrm{tg}(x^2+1)e^{-x}$	3,75	17,7	0,4
16	$\lvert x\rvert\sin(3x)$	$x^3\cos(x+2)$	$\sin x^2+x^{0.25}$	–5,55	10,33	0,1
17	$\lvert x\rvert^{2x+1}$	$\sin x^2$	$\ln^2\lvert x\rvert+\sqrt{x}$	–4,41	11,25	0,6
18	$\sin^2 x^3$	$\sqrt[5]{6x-x^2+1}$	$2\sin(x-e^{-x})$	2,17	19	0,8
19	$2xe^{-x}$	$(x-1)^3+\cos(x^3)$	$2\sqrt{x^3}\sin(x^3)$	8	12,1	0,11
20	$\ln(x^2+5)$	$\sin(e^x+2)$	$\mathrm{tg}(5x+1)$	–2,90	26,45	0,25
21	$2\sqrt{\lvert x^3\rvert}\sin(x^3)$	$(x+1)^2\cos x^3$	$\sqrt{x^4+2}+\sin x^2$	–1,25	67,32	0,3
22	$\cos x+x^3$	$\sqrt{x^3}\sin x$	$8+\cos(3x)$	–3,78	18,10	0,33

Продолжение таблицы 4.1

1	2	3	4	5	6	7
23	$x\sin(x+4)$	$\ln(4x^2+1)$	$\ln\sqrt[5]{5+x^2}$	–1,9	15,4	0,45
24	x^4+2x^3-x	$1{,}3\sqrt{4+x^2}$	$\lvert x+1\rvert^x$	–2,7	10,5	0,75
25	$\lvert x\rvert^5\operatorname{ctg}\lvert x\rvert$	$\ln(x^2+1)$	$e^{-2x}-\sqrt[3]{\lvert x+1\rvert}$	–3,8	17,1	0,65
26	$x^5\operatorname{ctg}(2x^3)$	$\sqrt[5]{x^4+3}$	$\lvert\sin^2 x+1\rvert^{2x}$	–4,6	29,9	0,3
27	$\operatorname{ctg}(3x-1)^2$	$2+xe^{-x}$	$\sin^3 x^2$	–5,74	19,2	0,4
28	$3x^5-\operatorname{ctg}(\pi x^3)$	$(x+1)^{0,3}+\sin 2x^3$	$5x-x^2$	–2,25	16,3	0,25
29	$\lvert x\rvert^{\sin(x)}+\sin(x)$	$3^{x+3}+2x$	$2^x+\sin(\pi x)$	–3,9	18,7	0,23
30	$x^2+\sin(7x)$	$\lvert x^3+10^x\rvert$	$\sqrt[7]{2x^4+x^2+1}$	–1,8	27,9	0,8

4.4 Контрольные вопросы

1 Методика построения графиков функций по их уравнениям (*для всех языков программирования*).

2 Можно ли использовать цикл `For` для построения графика функции? Если да, то как?

3 Вызов компонента `Chart` и его настройка.

4 Как построить нескольких графиков функций на одном компоненте `Chart`?

5 ЛАБОРАТОРНАЯ РАБОТА 5. ПОНЯТИЕ ОДНОМЕРНОГО МАССИВА. СЕЛЕКТИВНАЯ ОБРАБОТКА ЭЛЕМЕНТОВ МАССИВА

Цель: использование циклического оператора с параметром для ввода, вывода и обработки элементов массива; изучение методов селективной обработки элементов массива и нахождение минимального и максимального элемента в массиве; вывод элементов массива с помощью визуальных компонентов ***Visual Studio 2010***.

5.1 Теоретические сведения

5.1.1 Понятие одномерного массива

В языке программирования **C++** заложены средства для задания последовательностей упорядоченных данных. Такие последовательности называются *массивами*. *Массив* – это упорядоченная совокупность пронумерованных однотипных данных. В массивах должны быть упорядочены данные одного и того же типа. Номер элемента характеризуется целыми числами, называемыми *индексами.* **Массив характеризуется именем, размерностью и размером**.

В данной лабораторной работе будут рассматриваться массивы с целыми и вещественными типами данных, т. е. типы int, float или double.

Одномерный массив – это вектор (список связанных однотипных переменных). В частности, для векторов в приведенной записи индекс означает количество элементов. Для названия массива может быть использована переменная, состоящая из букв (буквы), букв с цифрами, букв с цифрами и знаком подчеркивания и т. д. в соответствии с правилами объявления переменных, принятых в языке **C++**. *Имя массива* образуется по общему правилу образования имен, т. е. представляет собой идентификатор, например, `A`, `B1`, `C8` и т. д. Однако оно не должно совпадать с именем ни одной простой переменной, используемой в той же программе.

Если размерность массива меньше, чем требуется, то компилятор не выдаст сообщения об ошибке. Выход за границы массивов должен следить только сам программист.

Общая форма записи одномерного массива:

```
тип имя_массива[размер];
```

В приведенной записи элемент *тип* объявляет базовый тип массива. Количество элементов, которые будут храниться в массиве с именем *имя_массива*, определяется элементом *размер*.

В языке **C++** индексация массива начинается с нуля. Например, если размер массива определен величиной 9, то в массиве можно хранить 10 элементов с индексацией 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Доступ к отдельному элементу массива осуществляется с помощью индекса. Индекс описывает позицию элемента внутри массива.

Индексы определяют положение элемента в массиве. *Число индексов определяет размерность массива*, т. е. форму его компоновки: одномерный, двумерный и т. д. Одномерный массив соответствует линейной таблице. Его элемент обозначается переменной с одним индексом: `A[0]`, `A[i]`, соответственно, первый и *i*-й элементы одномерного массива A.

Все массивы занимают смежные ячейки памяти, т. е. элементы массива в памяти расположены последовательно друг за другом. Ячейка памяти с наименьшим адресом относится к первому элементу массива, а с наибольшим – к последнему.

Для одномерных массивов общий размер массива в байтах вычисляется по формуле:

```
всего байт = размер типа в байтах * количество элементов.
```

В языке **C++** нельзя присвоить один массив другому. Для передачи элементов одного массива другому необходимо выполнить присвоение поэлементно.

5.1.2 Инициализация массива

В языке **C++** массив при объявлении можно инициализировать.

Общая форма инициализации массива:

```
тип имя_массива[размер1] * [размерN] = {список_значений};
```

В *список_значений* входят константы, разделенные запятыми. Типы констант должны быть совместимыми с типом массива.

Пример инициализации одномерного массива:

```
int A[5] = {1, 2, 3, 4, 5};
```

При этом A[0] = 1; A[1] = 2 и т. д.

В языке **C** возможна инициализация безразмерных массивов. Например, для одномерного массива:

```
int A[ ] = {1, 2, 3, 4, 5};
```

5.1.3 Селективная обработка элементов массива

Селективная обработка массива – это выделение из массивов элементов, удовлетворяющих условию, и обработка выделенных фрагментов. Часто из выделенных фрагментов формируют новый (рабочий) массив, который далее и обрабатывают.

Наиболее часто встречаются такие условия обработки элементов массива:

– четные	A[i] %2 ==0
– нечетные	A[i] %2 !=0
– кратные k	A[i] % k ==0
– некратные k	A[i] % k !=0
– стоящие на четных местах	(i+1) % 2==0
– стоящие на нечетных местах	(i+1) % 2!=0
– положительные	A[i] >0
– отрицательные	A[i] <0
– в интервале (x1,x2)	((A[i]>x1) && (A[i]<x2)).

5.2 Пример выполнения работы

Создать Windows-приложение, которое предлагает пользователю задать размер линейного массива, заполняет автоматически этот массив случайными целыми числами в диапазоне от –10 до 10, выводит элементы массива, затем по выбору пользователя определяет сумму четных элементов массива и количество положительных элементов массива.

Ход выполнения

1 Войти в среду ***Visual Studio 2010.***

2 В окне **Создать Проект** следует развернуть узел `Visual C++`, обратиться к пункту `CLR` и на центральной панели выбрать *Приложение `Windows Form`.*

3 Затем в поле редактора **Имя** (где по умолчанию имеется <Введите имя>) следует ввести имя проекта, **Visual_Lab5**. В поле **Расположение** можно указать путь размещения проекта или выбрать путь размещения проекта с помощью клавиши (кнопки) **Обзор** (например, N:\\CI\2_trim\Lab5).

4 Для формы изменить значение свойства **`Text`,** занеся следующие данные: ***«Выполнил студент группы ЭСА-11-1 Иванов П. А. Лабораторная работа 5».***

5 Свойству **`FormBorderStyle`** формы (стиль рамки окна) присвоить значение ***`FixedToolWindow`***. Это значение определяет окно как диалоговое, его размеры на этапе прогона приложения (в процессе работы приложения) не могут быть изменены.

6 В верхней части окна разместить контейнер **`GroupBox1`** и в свойстве **`Text`** занести текст «***Ввод элементов массива***».

7 На этом же контейнере **`GroupBox1`** разместить пять компонентов: **`Label1`**, **`Label2`**, **`TextBox1`**, **`TextBox2`** и **`Button1`**. Для первого компонента **`Label1`** в свойство **`Text`** занести текст «***Введите число элементов масси-***

ва». Для второго компонента **Label2** в свойство **Text** занести текст «***Исходный массив***». Для компонента **TextBox2** в свойстве **ReadOnly** (только чтение) присвоить значение ***true***, запрещающее пользователю заносить в компонент какие-либо данные. Для компонента **TextBox1** данные будут вводиться с клавиатуры с помощью компонента **TextBox.** В свойства **Text** этого компонента ввести какое-либо значение – значение по умолчанию. Это значение будет показываться при запуске приложения на выполнение. При выполнении приложения его можно будет заменить другим.

8 Выделить компонент **Button1** и в свойство **Text** занести текст «***Создать массив***». При нажатии на эту кнопку будет автоматически создаваться массив с помощью генератора случайных чисел, элементы которого будут в указанном диапазоне [–10; 10] выведены в компоненте **TextBox2**. Получим следующую форму (рис. 5.1).

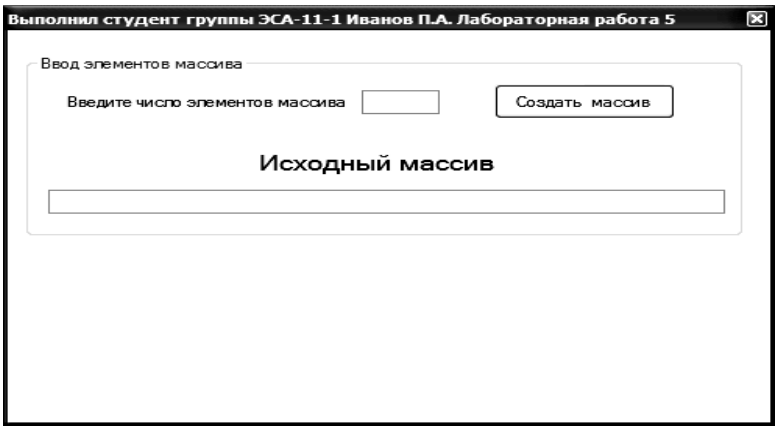


Рисунок 5.1 – Окно формы на этапе создания программы

9 В нижней части окна разместить контейнер **GroupBox2** и в свойстве **Text** занести текст «***Найти***».

10 На этом же контейнере **GroupBox2** разместить шесть компонентов: **TextBox3**, **TextBox4**, **checkBox1**, **checkBox2**, **Button2** и **Button3.**

11 Выделить компонент **checkBox1** и в свойство **Text** занести текст «***сумму четных элементов массива***». Аналогично для компонента **checkBox2** в свойство **Text** занести текст «***количество положительных элементов массива***».

12 Выделить компонент **Button2** и в свойство **Text** занести текст «***Вычислить***». Аналогично для компонента **Button3** в свойство **Text** занести текст «***Выход***».

13 Компоненты **TextBox3** и **TextBox4** будут использованы для вывода результатов нахождения задач, поэтому необходимо запретить ввод в них данных пользователем. Для этого свойству **ReadOnly** (только чтение) присвоить значение ***true***, запрещающее пользователю заносить в компонент какие-либо данные.

14 В результате проведенных операций должна получиться форма примерно такого вида, как показано на рис. 5.2.

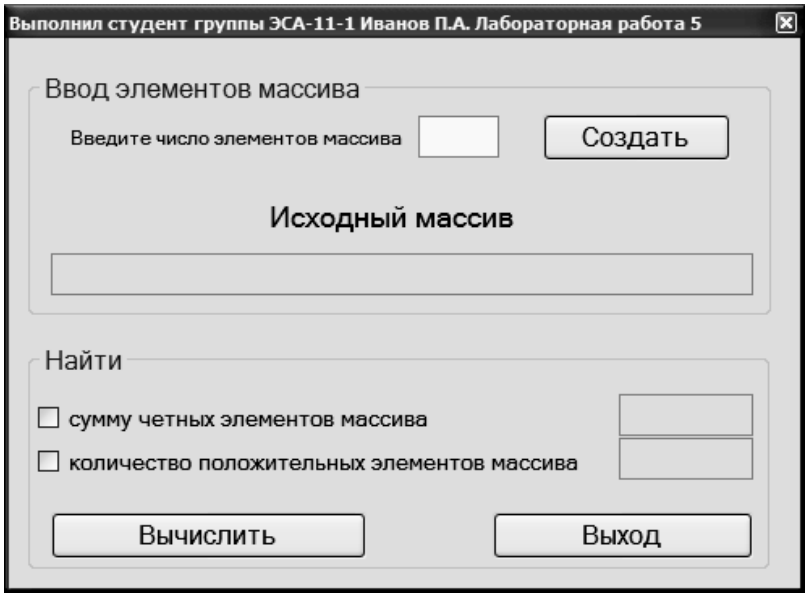


Рисунок 5.2 – Окно формы программы

15 Перейти к коду программы и после строки

```
#pragma once
```

подключить библиотеки использования математических функций, описание переменных и размера массива, для этого вставить следующие строки:

```
#include<cmath>
#include <stdlib.h>
#define m 15
int i,n;
int A[m];
```

16 Создать событие ***Click*** для кнопки **Button1** с надписью ***«Создать»***. Для этого сделать двойной щелчок левой кнопкой мыши по компоненту **Button1**. Внести изменения в код подпрограммы:

//Проверка, что не пустой компонент textBox1

```
if(textBox1->Text!="")
{n=Convert::ToInt32(textBox1->Text); }else
{MessageBox::Show( "Заполните, пожалуйста, данные", "Ошибка
ввода данных", MessageBoxButtons::OK, MessageBoxI-
con::Exclamation );}
```

//очистка компонента textBox2

```
{textBox2->Text = "";
```

//Процесс создания массива и заполнение компонента textBox2 случайными числами из диапазона [-10;10]

```
for (i = 0; i < n; ++i)
{
A[i] = rand ( ) % 21-10;
this->textBox2->AppendText(A[i]+"  ");
}
}
```

17 Создать событие ***Click*** для кнопки **Button2** с надписью ***«Вычислить»***. Для этого сделать двойной щелчок левой кнопкой мыши по компоненту **Button2** и в полученную заготовку подпрограммы вставить код для расчетов:

```
//Обнуляем переменные kol и sum
int kol=0; int sum=0;
     for (i = 0; i < n; ++i)
{
//Проверяем условие и находим сумму
if (A[i]%2==0){sum=sum+A[i];}
//Проверяем условие и находим количество
if (A[i]>0){kol=kol+1;}
}
//Если компоненты checkBox включены, то выводим результаты заданий в компоненты textBox
if (checkBox1->Checked==true){this->textBox3->Text=Convert::ToString
(sum);}
if (checkBox2->Checked==true){this->textBox4->Text=Convert::ToString
(kol);}
```

18 На примере лабораторной работы 2 сделать проверку на корректность ввода данных (количество элементов массива), используя событие **Leave** в окне свойств (в код программы добавить 1 процедуру, проверяющую правильность ввода).

19 Создать событие ***Click*** для кнопки **Button3** с надписью ***«Выход»***. Для этого сделать двойной щелчок левой кнопкой мыши по компоненту **Button3** и в полученную заготовку подпрограммы вставить код выхода из программы (уже использовался в предыдущих лабораторных работах).

20 Запустить программу на выполнение, нажав на функциональную кнопку **F5**. Получим следующий вид окна рис. 5.3.

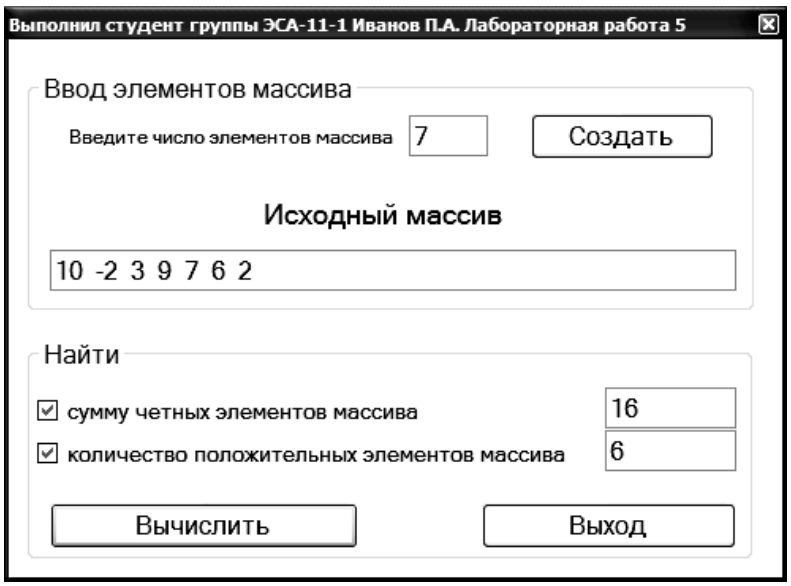


Рисунок 5.3 – Рабочий вид формы приложения

5.3 Рабочее задание

Создать Windows-приложение, которое предлагает пользователю задать размер линейного массива, заполняет автоматически этот массив случайными целыми числами в диапазоне от –50 до 70, выводит элементы этого массива, затем по выбору пользователя определяет, соответственно, для каждого варианта задание **а), б), в).**

Данные взять из таблицы 5.1.

Таблица 5.1 – Индивидуальные задания

Вариант	**задание**	**Условие задания**
1	2	3
1	***а***	Найти количество отрицательных элементов
	б	Найти сумму отрицательных элементов
	в	Найти минимальный элемент кратный пяти
2	***а***	Найти количество четных элементов
	б	Найти сумму элементов кратных 3
	в	Найти разность максимального и минимального элементов массива
3	***а***	Найти среднее арифметическое элементов массива
	б	Найти сумму наибольшего и наименьшего элементов массива
	в	Найти максимальный по модулю элемент массива
4	***а***	Вычислить среднее арифметическое четных элементов массива
	б	Найти сумму минимального положительного элемента массива и его номера
	в	Найти произведение модулей наибольшего отрицательного и наименьшего четного элементов массива
5	***а***	Найти количество элементов, кратных 5
	б	Найти сумму четных элементов массива, стоящих на нечетных местах
	в	Найти сумму второго и наибольшего положительного элементов массива
6	***а***	Найти среднее геометрическое четных элементов массива
	б	Найти номер наибольшего по модулю элемента массива
	в	Найти максимальный четный элемент массива
7	***а***	Вычислить среднее арифметическое максимального и минимального элементов массива
	б	Найти минимальный по модулю элемент массива
	в	Найти сумму элементов из интервала [0; 10]
8	***а***	Вычислить среднее геометрическое номеров максимального и минимального элементов массива
	б	Найти разность суммы положительных и произведения отрицательных чисел массива
	в	Найти количество положительных элементов
9	***а***	Найти сумму отрицательных элементов
	б	Найти сумму элементов массива, у которых индекс кратен 3
	в	Найти произведение модулей наибольшего и наименьшего элементов массива

Продолжение таблицы 5.1

1	2	3
10	***а***	Найти сумму отрицательных элементов
	б	Найти сумму второго и наибольшего отрицательного элементов массива
	в	Найти разность максимального и минимального элементов массива
11	***а***	Вычислить среднее арифметическое четных элементов массива
	б	Найти сумму элементов в диапазоне [–10; 0]
	в	Найти максимальный по модулю элемент массива
12	***а***	Найти сумму минимального положительного элемента массива и его номера
	б	Найти сумму четных элементов массива из диапазона [–20; 30]
	в	Найти минимальный по модулю элемент массива
13	***а***	Найти максимальный по модулю элемент
	б	Найти среднее арифметичекое элементов массива
	в	Найти сумму отрицательных элементов
14	***а***	Найти количество элементов, кратных 4
	б	Найти сумму отрицательных элементов
	в	Найти сумму наибольшего и наименьшего элементов массива
15	***а***	Найти разность максимального и минимального положительных элементов
	б	Найти сумму нечетных элементов
	в	Найти минимальный элемент из диапазона [–20; 30]
16	***а***	Найти количество положительных элементов
	б	Найти сумму элементов больших 3
	в	Найти максимальный элемент массива
17	***а***	Найти количество отрицательных элементов
	б	Найти сумму отрицательных элементов
	в	Найти минимальный элемент, кратный пяти
18	***а***	Найти количество четных элементов
	б	Найти сумму элементов, кратных 3
	в	Найти разность максимального и минимального элементов массива
19	***а***	Найти среднее арифметичекое элементов массива
	б	Найти сумму наибольшего и наименьшего элементов массива
	в	Найти максимальный по модулю элемент массива
20	***а***	Вычислить среднее арифметическое четных элементов массива
	б	Найти сумму минимального положительного элемента массива и его номера
	в	Найти произведение модулей наибольшего отрицательного и наименьшего четного элементов массива
21	***а***	Найти количество элементов, кратных 5
	б	Найти сумму четных элементов массива, стоящих на нечетных местах
	в	Найти сумму второго и наибольшего положительного элементов массива
22	***а***	Найти среднее геометрическое четных элементов массива
	б	Найти номер наибольшего по модулю элемента массива
	в	Найти максимальный четный элемент массива

Продолжение таблицы 5.1

1	2	3
23	***а***	Вычислить среднее арифметическое максимального и минимального элементов массива
	б	Найти минимальный по модулю элемент массива
	в	Найти сумму элементов из интервала [0; 10]
24	***а***	Вычислить среднее геометрическое номеров максимального и минимального элементов массива
	б	Найти разность суммы положительных и произведения отрицательных чисел массива
	в	Найти количество положительных элементов
25	***а***	Найти сумму отрицательных элементов
	б	Найти сумму элементов массива, у которых индекс кратен 3
	в	Найти произведение модулей наибольшего и наименьшего элементов массива
26	***а***	Найти сумму отрицательных элементов
	б	Найти сумму второго и наибольшего положительного элементов массива
	в	Найти разность максимального и минимального элементов массива
27	***а***	Вычислить среднее арифметическое четных элементов массива
	б	Найти сумму элементов в диапазоне [–10; 20]
	в	Найти максимальный по модулю элемент массива
28	***а***	Найти сумму минимального положительного элемента массива и его номера
	б	Найти сумму четных элементов массива из диапазона [–20; 30]
	в	Найти минимальный по модулю элемент массива
29	***а***	Найти количество элементов, кратных 5
	б	Найти сумму четных элементов массива, стоящих на нечетных местах
	в	Найти максимальный четный элемент массива
30	***а***	Найти количество положительных элементов
	б	Найти сумму элементов больших 3
	в	Найти максимальный элемент массива

5.4 Контрольные вопросы

1 Понятие одномерного числового массива в языке **C**++?

2 Как организуется индексирование числовых массивов в языке **C**++?

3 Для чего применяется начальная инициализация числовых массивов при дальнейшем их использовании?

4 Как использовать генератор случайных чисел для заполнения массива элементами?

5 Условия селективной обработки элементов массива.

6 Нахождение минимального и максимального элементов массива.

7 С помощью каких визуальных компонентов можно сделать ввод элементов массива в ручном режиме?

6 ЛАБОРАТОРНАЯ РАБОТА 6. МНОГОМЕРНЫЙ МАССИВ. ПОНЯТИЕ МАТРИЦЫ. СЕЛЕКТИВНАЯ ОБРАБОТКА ЭЛЕМЕНТОВ СТРОК, СТОЛБЦОВ И ДИАГОНАЛЕЙ МАТРИЦЫ

Цель: изучение использования вложенных циклических операторов для ввода, вывода и обработки элементов матрицы; обработка элементов столбцов, строк и диагоналей матрицы. Использование визуального компонента `DataGridView` для работы с матрицами у ***Visual Studio 2010***.

6.1 Теоретические сведения

6.1.1 Понятие матрицы

Двумерный массив (матрица) — это упорядоченная совокупность пронумерованных однотипных данных. **Матрица характеризуется именем, размерностью и размером**.

Для *имени* матрицы может быть использована переменная, состоящая из букв (буквы), букв с цифрами, букв с цифрами и знаком подчеркивания и т. д. в соответствии с правилами объявления переменных, принятых в языке **C++**. *Имя матрицы* образуется по общему правилу образования имен, т. е. представляет собой идентификатор, например, `A`, `Bl`, `C_8` и т. д. Однако оно не должно совпадать с именем ни одной простой переменной, используемой в той же программе.

В матрице позиция любого элемента определяется двумя целыми числами, называемыми *индексами.* Индексы каждого из размеров массива начинаются с 0 (с нуля). Число индексов определяет *размерность* массива *(у матрицы два индекса: первый показывает номер строки, а второй номер столбца, где элемент находится).*

Двухмерный массив (*матрица*) представляет собой список одномерных массивов.

Общая форма записи двухмерного массива:

```
тип имя_массива[размер1] [размер2];
```

В приведенной записи *размер1* означает количество строк двухмерного массива, а *размер2* – количество столбцов.

Место хранения для всех элементов матрицы определяется во время компиляции. Память, выделенная для хранения массива, используется в течение всего времени существования массива.

Для двухмерных массивов общий размер массива в байтах вычисляется по формуле:

```
всего байт = число строк * число столбцов * размер типа в байтах.
```

Для определения размера типа в байтах применяется функция `sizeof()`, которая возвращает целое число. Например, `sizeof (float)`.

6.1.2 Инициализация матрицы

При инициализации матрицы для улучшения наглядности элементы инициализации каждого измерения можно заключать в фигурные скобки.

Пример инициализации двухмерного массива:

```
int  MN[3][4] = {
{1, 2, 3, 4},
{5, 6, 7, 8},
{9, 10, 11, 12}
};
```

Массив MN[3][4] – это матрица, у которой 3 строки и 4 столбца.

В языке **C++** нельзя присвоить одну матрицу другой. Для передачи элементов одной матрицы другой необходимо выполнить присвоение поэлементно.

Для доступа к элементу матрицы следует указать имя матрицы с последующим числом (индексом), заключенным в квадратные скобки.

Элементы массива можно использовать в любом выражении точно также, как и значение константы или переменной.

Например:

```
a[0][0]=11.2;
a[1][2]=10.2;
a[3][1]=22.1;
a[4][2]=1.1;
Y = 2*a[0][1] – a[1][0];
```

6.1.3 Селективная обработка элементов матрицы

Селективная обработка матрицы – это выделение из матрицы элементов, удовлетворяющих условию, и обработка выделенных фрагментов. Часто из выделенных фрагментов формируют новый (рабочий) массив, который далее и обрабатывают.

Наиболее часто встречаются такие условия обработки элементов массива:

– четные	`A[i] %2 ==0`
– нечетные	`A[i] %2 !=0`
– кратные k	`A[i] % k ==0`
– не кратные k	`A[i] % k !=0`

```
– положительные                    A[i] >0
– отрицательные                    A[i] <0
– в интервале [x1,x2]              ((A[i]>=x1)&&(A[i]<=x2))
```

При обработке матриц часто приходится выделять элементы:

```
– k-й строки            A[i][j], где i==k-1, j=0,…, M-1
– k-го столбца          A[i][j], где i=0, …, N-1; j==k-1
```

а для квадратных матриц (M=N) также:

```
– главной диагонали          i==j,
– побочной диагонали         j=N-1-i,
– наддиагональные            j>i,
– поддиагональные            j<i,
```

где i – номер строки, j – номер столбца.

Например, для нахождения условия «найти элементы 3 столбца 2 строки», укажем условие с помощью условного `IF`:

```
If (i==1&&j==2) {действие};
```

6.1.4 Использование элемента управления **DataGridView** *для работы с матрицами*

`DataGridView` – это элемент управления, который может отображать буквально любой вид данных в ячейках прямоугольной сетки.

Элемент управления `DataGridView` позволяет отображать и изменять прямоугольный массив данных из множества различных источников данных. Его можно использовать также для отображения практически любых данных, созданных непосредственно в программе. По своей сущности это сложный элемент управления, который обеспечивает огромную гибкость при его применении, и преимуществами множества его функциональных возможностей можно воспользоваться через множество свойств, функций и событий. В то же время применение элемента управления `DataGridView` может быть поразительно простым. Можно не обращать внимания на внутреннюю сложность и использовать его посредством инструмента `Form Design` (Конструктор форм), которое берет на себя заботу обо всех основных характеристиках.

Данные элемента управления `DataGridView` отображаются в прямоугольном массиве ячеек, которые можно представить в виде коллекции строк или столбцов. Каждый столбец ячеек имеет в верхней части ячейку заголовка, которая, как правило, содержит идентифицирующий этот столбец текст, а в начале каждой строки расположена ячейка заголовка строки, как показано на рис. 6.1. Обращение к строкам и столбцам ячеек выполняется через свойства объекта `DataGridView`. Свойство `Rows` возвращает значение типа `DataGridRowCollection`, представляющее коллекцию всех строк, а обращение к конкретной строке выполняется с помощью индекса, что можно видеть на рис. 6.1.

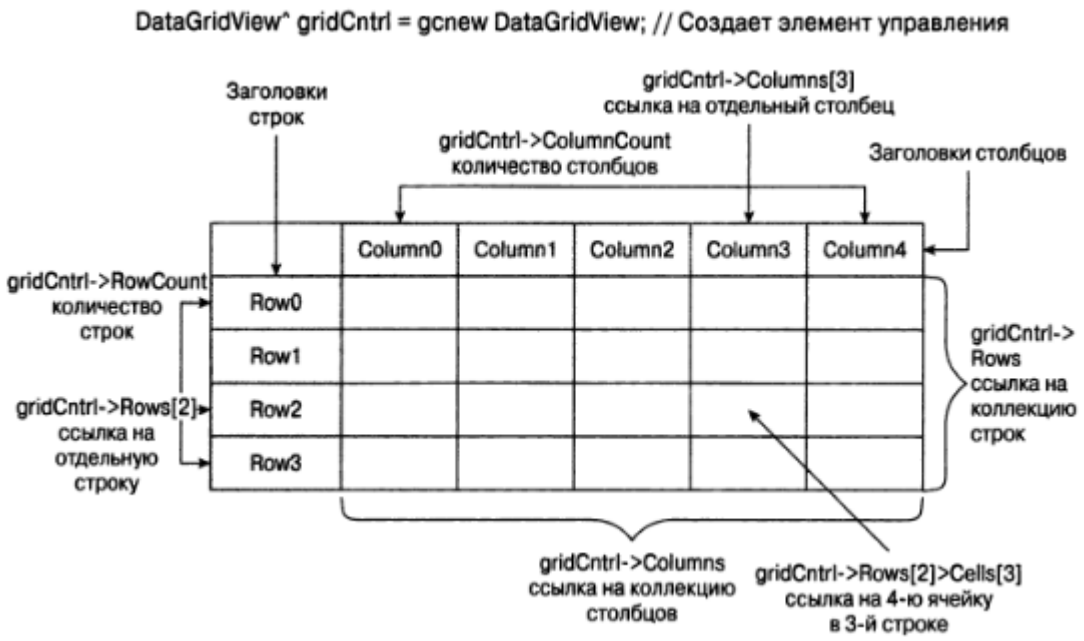


Рисунок 6.1 – Компонент `DataGridWiew`

Аналогично свойство `Columns` элемента управления возвращает значение типа `DataGridViewColumnCollection`, которое также можно индексировать для обращения к конкретному столбцу. Индексация строк и столбцов осуществляется, начиная с нуля. Свойство `Cells` объекта `DataGridRowCollection` представляет коллекцию, содержащую ячейки строки, и это свойство можно индексировать для получения доступа к конкретной ячейке в строке. Пример ссылки на четвертую ячейку в третьей строке приведен на рис. 6.1.

Количество строк доступно как значение свойства `RowCount`, а свойство `ColumnCount` возвращает количество столбцов. Вначале, когда элемент управления еще нс связан с источником данных, он не будет содержать ни строк, ни столбцов. Количество столбцов и / или строк можно определить, устанавливая значения свойств элемента управления, но при его использовании для отображения данных из источника данных это действие выполняется автоматически.

Элемент управления `DataGridView` применяется в трех различных режимах. Мы будет применять «*несвязанный режим*». В несвязанном режиме передача данных элементу управления выполняется вручную, как правило, с помощью функции `Add ()` применительно к свойству Rows элемента управления. Этот режим следует использовать для отображения сравнительно небольших объемов данных.

В несвязанном режиме элемент управления `DataGridView` можно использовать для отображения в приложении любых данных, которые могут быть отображены в табличном виде. Потому этот инструмент очень удобен для отображения данных во множестве разнообразных приложений.

6.1.4.1 Использование элемента управления DataGridView в несвязанном режиме. Данные в элементе управления `DataGridView` хранятся в прямоугольной структуре, определяемой свойствами `Rows` и `Columns` элемента управления. В несвязанном режиме добавление данных в элемент управления выполняется с помощью функции `Add` () применительно к свойству `Rows`. Но прежде чем в элемент управления можно будет добавлять строки, потребуется определить столбцы – задать количество элементов в строке. Программная установка свойства `ColumnCount` элемента управления определяет количество столбцов и указывает, что элемент управления будет работать в несвязанном режиме. Следующие операторы создают элемент управления, обращение к которому выполняется посредством дескриптора `DataGridView`, а затем устанавливают количество столбцов, равное 3:

```
DataGridView^ dataGridView = dcnew DataGridView;
DataGridView->ColumnCount = 3; // Устанавливает количество столбцов.
```

При желании столбцы в элементе управления можно пометить путем установки свойства `Name` для каждого столбца, указывая заголовки, идентифицирующие данные в каждом из них. Это можно было бы выполнить так:

```
dataGridView->Columns[0]->Name="Name";
dataGrldView->Columns[l]->Name="Phone Number";
dataGridView->Columns[2]->Name="Address";
```

Свойство `Columns` элемента управления – индексированное свойство, поэтому доступ к отдельным столбцам можно получить с помощью значений индексов, начинающихся с 0. Таким образом, эти три оператора снабжают метками три столбца в элементе управления `DataGridView`. Заголовки столбцов можно определить также в окне `Properties` (Свойства) элемента управления.

Свойство Rows возвращает значение, представляющее собой коллекцию типа **`DataGridViewRowCollection`**, который определен в пространстве имен `System:: Windows: :Forms`. Свойство `Count` упомянутой коллекции возвращает количество строк. Кроме того, коллекция имеет также свойство начальной индексации, возвращающее строку в позиции данного индекса. Коллекция строк обладает множеством функций, наиболее полезных, предназначенных для добавления и удаления строк (табл. 6.1).

Таблица 6.1 – Свойства для добавления и удаления строк

Функции	Описание
`Add ()`	Добавляет в коллекцию одну или более строк
`Insert ()`	Вставляет в коллекцию одну или более строк
`Clear()`	Удаляет все строки коллекции
`AddCopy()`	Добавляет копию строки, указанную в аргументе
`InsertCopy()`	Вставляет копию строки, указанную первым аргументом, в позицию, которая задана вторым аргументом
`Remove()`	Удаляет строку, указанную аргументом типа `DataGridviewRow`.
`RemoveAt ()`	Удаляет строку, указанную значением индекса, которое передано в качестве аргумента.

Функция Add (), применяемая к значению, возвращенному свойством Rows, существует в виде четырех перегруженных версий, которые позволяют добавлять в элемент управления строку данных.

Все версии функции Add () возвращают значение типа int, представляющее собой индекс последней добавленной в коллекцию строки. Если значение свойства SataSource элемента управления DataGridView не нулевое или элемент управления не содержит столбцов, все версии функции Add () генерируют исключение типа System:: InvalidOperationException.

Таблица 6.2 – Перегруженные версии функции Add () коллекции DataGridViewRowCollection

Функция	Описание
Add ()	Добавляет в коллекцию одну новую строку
Add(int rowCount)	Добавляет в коллекцию rowCount новых строк. Если значение rowCount нулевое или отрицательное, функция генерирует исключение типа System::ArgumentOutOfRangeException
Add(DataGridViewRow^ row)	Добавляет строку, указанную аргументом. Объект DataGridviewRow содержит коллекцию ячеек строки, а также параметры, которые определяют внешний вид ячеек в строке
Add(... Object^ object)	Добавляет новую строку и заполняет ячейки строки объектами, указанными аргументами

6.1.4.2 Персональная настройка элемента управления DataGridView. Внешний вид каждой ячейки в элементе управления DataGridView определяется объектом типа DataGridViewCellStyle, имеющим свойства, описанные в табл. 6.3.

Таблица 6.3 – Свойства объекта DataGridViewCellStyle, влияющие на внешний вид ячейки

Свойство	Описание
1	2
BackColor	Это значение – объект System::Drawing::Color. который определяет цвет фона ячейки. В классе Color диапазон стандартных цветов определен в виде статических членов. Значение, используемое по умолчанию — Color:: Empty
ForeColor	Это значение – объект color, который определяет цвет изображения ячейки. Значение, используемое по умолчанию – Color::Empty
SelectionBackColor	Это значение – объект color, который определяет цвет фона выбранной ячейки. Значение, используемое по умолчанию – Color::Empty

Продолжение таблицы 6.3

1	2
SelectlonForeColor	Это значение – объект color, который определяет цвет изображения выбранной ячейки. Значение, используемое по умолчанию – Color::Empty
Font	Это значение – объект System::Drawing::Font, определяющий шрифт, который должен использоваться для отображения текста е ячейке. Значение, используемое по умолчанию – null
Alignment	Это значение определяет выравнивание содержимого ячейки. Допустимые значения определены перечислением DataGridViewAlignnent, поэтому значение может быть любой из следующих констант; Bottomcenter, BottomLeft, BottomRight, MiddleCenter, MiddleLeft, MiddleRight, TopCenter, TopLeft, TopRight. NotSet. Значение, используемое по умолчанию – NotSet
WrapMode	Это значение определяет переход текста в ячейке в следующую строку, если он слишком длинен, чтобы уместиться в ячейке. Допустимое значение – одна из констант, определенных перечислимым объектом DataGridViewTriState: ***True, False, NotSet***. Значение, используемое по умолчанию – NotSet
Padding	Это значение – Объект типа System::Windows::Forms::Padding, который определяет пробел между содержимым и краем ячейки. Конструктор класса Padding требует передачи аргумента типа int, который представляет значение дополнения содержимого ячейки пробелами, измеренное в пикселях. Значение, используемое по умолчанию, соответствует отсутствию дополнения содержимого ячейки
Format	Это значение – строка формата, которая определяет способ форматирования содержимого строки. Это форматирование аналогично используемому в функции Console::WriteLine(). Значение, используемое по умолчанию, – пустая строка

Приведенный в табл. 6.3 список включает далеко не все свойства объекта DataGridViewCellStyle, а лишь те, которые связаны с внешним видом ячейки.

Определение внешнего вида конкретной ячейки – достаточно сложная задача, поскольку в элементе управления DataGridView можно устанавливать множество различных свойств, определяющих способ отображения данной ячейки или группы ячеек, причем некоторые из этих свойств могут действовать в любой заданный момент времени. Например, можно определить значения свойств, которые указывают внешний вид строки или столбца ячеек или всех ячеек в элементе управления, причем все эти свойства могут действовать одновременно. Очевидно, что поскольку строка и столбец всегда пересекаются, все три значения свойств применимы к любой отдельной ячейке, что ведет к возникновению явного конфликта.

Каждая ячейка в элементе управления DataGridView представлена объектом System::Windows::Forms::DataViewCell, а внешний вид каждой конкретной ячейки, включая ячейки заголовков, определяется значением ее свойства InheritedStyle. Это значение для данной ячейки определяется путем просмотра всех доступных свойств, которые возвращают значение, являющееся объектом DataGridViewStyle примененного к ячейке, и последующего упорядочения этих значений по приоритету. Значение с наивысшим приоритетом будет установлено в качестве действующего. Определение значения свойства InheritedStyle для ячеек заголовков строк и столбцов выполняется иначе, чем для остальных ячеек, поэтому они будут рассматриваться отдельно, начиная с ячеек заголовков.

6.1.4.3 Настройка ячеек заголовков. Значение свойства InheritedStyle для каждой ячейки заголовка в элементе управления определяется учетом значений свойств в представленной ниже последовательности:

– свойства Style объекта DataGridViewCell, который представляет ячейку;

– свойства ColumnHeaderDefaultCellStyle или RowHeadersDefaultCellStyle объекта элемента управления;

– свойства DefaultCellStyle объекта элемента управления.

Таким образом, если значение свойства Style объекта ячейки установлено, свойству InheritedStyle ячейки присваивается это значение, которое и определяет ее внешний вид. Если это значение не установлено, в действие вступает следующее значение, если оно установлено. Если и второе свойство не установлено, применяется свойство DefaultCellStyle элемента управления.

Следует помнить, что значение свойства InheritedStyle – объект типа DataGridViewCellStyle, который сам обладает свойствами, определяющими различные аспекты внешнего вида ячейки. Процесс учета приоритета применяется к каждому из свойств объекта DataGridViewCellStyle, поэтому общая последовательность приоритетов может включать в себя более одного свойства.

6.1.4.4 Настройка ячеек, не являющихся заголовками. Значение свойства InheritedStyle каждой ячейки элемента управления, не являющееся заголовком (то есть содержащее данные) определяется свойствами объекта DataGridView в описанной далее последовательности:

– свойством Style объекта DataGridViewCell, который представляет ячейку;

– свойством DefaultCellStyle объекта DatagridViewRow, который представляет строку, содержащую ячейку. Как правило, ссылку на объект

DatagridViewRow нужно будет выполнять посредством индексации свойства Rows объекта элемента управления;

– свойством AlternatingRowsDefaultCellStyle объекта элемента управления. Это свойство применяется только к ячейкам строк с нечетными номерами индексов;

– свойством RowsDefaultCellStyle объекта элемента управления;

– свойством DefaultCellStyle объекта DataGridViewColumn, содержащего ячейку. Как правило, обращение к объекту DataGridViewColumn будет выполняться посредством индексации свойства Columns объекта элемента управления;

– свойством DefaultCellstyle объекта элемента управления.

Вообще говоря, для каждой ячейки можно было бы использовать отдельный объект DataGridViewCellStyle, но для повышения эффективности количество таких объектов следует сохранять минимальным.

6.1.4.5 Настройка элемента управления. Нам необходимо, чтобы элемент управления размещался в фиксированной позиции в клиентской области формы. Это можно осуществить, устанавливая значение свойства Dock:

```
dataGridView->Dock = DockStyle::Fill;
```

В качестве значения свойства Dock должна быть установлена одна из констант, определенных перечислением DockStyle. Другими допустимыми значениями этого свойства являются Top (Вверху), Bottom (Внизу), Left (Слева), Right (Справа) и None (Нет), которые указывают стороны элемента управления, привязанные к позиции размещения.

Привязку позиции элемента управления к клиентской области формы можно выполнять также посредством установки свойства Anchor элемента управления. Значение этого свойства указывает края элемента управления, которые должны быть привязаны к клиентской области формы. Значение является побитовой комбинацией констант, определенных перечислением AnchorStyles, и может принимать любые или все из значений Top, Bottom, Left и Right. Например, чтобы привязать верхнюю и левую стороны элемента управления, в качестве значения нужно было бы указать AnchorStyles::Top & AnchorStyles::Left. Определение свойства Anchor ведет к фиксации позиции элемента управления и его линеек прокрутки внутри контейнера заданного размера. Поэтому при изменении размера окна приложения элемент управления и его линейки прокрутки сохраняют свои размеры. Если установить значение свойства Dock так, как в предыдущем операторе, при изменении размера окна приложения отображаемая в нем часть элемента управления будет изменяться с соответствующим изменением линеек прокрутки. Поэтому теперь работать с приложением значительно удобнее.

Нам требуется, чтобы ширина столбцов автоматически изменялась, обеспечивая отображение полных строк данных в ячейках. Для этого можно вызывать функцию `AutoResizeColumns()`:

```
dataGridview->AutoResizeColumns();
```

Этот оператор подбирает ширину всех столбцов в соответствии с текущим содержимым, в том числе в соответствии с содержимым ячеек заголовков. Обратите внимание, что эта настройка выполняется во время вызова функции, поэтому к моменту её вызова содержимое уже должно существовать. При последующем изменении содержимого ширина столбца не изменяется. Если требуется, чтобы ширина столбца автоматически изменялась при каждом изменении содержимого ячеек, для элемента управления нужно установить также свойство `AutoSizeColumnsMode`:

```
dataGridView-
>AutoSizeColumnsMode=DataGridViewAutoSizeColumnsMode:
:AllCells;
```

Значением этого свойства должна быть одна из констант, определенных перечислением `DataGridViewAutoSizeColumnsMode`. Другими возможными значениями являются `ColumnHeader` (Заголовок столбца), `AllCellsExceptHeader` (Все ячейки, кроме заголовка), `DisplayedCells` (Отображаемые ячейки), `DisplayedCellsExceptHeader` (Отображаемые ячейки, кроме заголовка). `Fill` (Заполнение) и `None` (Нет). Естественно, эти значения отображаются также в списке значений этого свойства на странице *`Properties`* (свойства) элемента управления `DataGridView`.

В некоторых ситуациях требуется, чтобы автоматический подбор ширины при изменении содержимого выполнялся только для определенных столбцов. В этом случае значение свойства `AutoSizeMode` необходимо устанавливать для объекта столбца.

Существует еще две перегруженных версии функции `AutoSizeColumns()`. Одна принимает аргумент типа `DataGridViewAutoSizeColumnsMode`, определяющий ячейки, на которые функция оказывает воздействие. Вторая перегрузка является защищенной и, следовательно, предназначена для использования в производном классе. Она принимает дополнительный аргумент типа `bool`, который указывает, должна ли высота ячейки учитываться при вычислении новой ширины.

Используемый по умолчанию цвет фона всех ячеек элемента управления можно установить следующим образом:

```
dataGridview->DefaultCellStyle->Backcolor=color::Pink;
```

Этот оператор устанавливает в качестве цвета фона стандартный цвет `Pink` (Розовый), который определен в качестве статического члена класса `Color` (Цвет). Свойства `DefaultCellStyle` объекта элемента управления определяют только те атрибуты стиля, которые применяются к ячейке при отсутствии какого-либо другого действующего стиля ячейки, имеющего более высокий приоритет.

Можно также определить применяемый по умолчанию цвет изображения всех ячеек:

```
dataGridView->DefaultCellStyle->ForeColor=Color::DarkBlue;
```

Для визуального определения выбранных ячеек можно указать цвета изображения и фона выбранных ячеек. Эти цвета можно было бы определить следующим образом:

```
dataGridView->DefaultCellStyle->SelectionBackColor=Color:
:Green;
```

Естественно, смысл программного определения свойств в том, что установленные значения применяются во время выполнения, что позволяет определять значения в зависимости от условий и значений данных, существующих во время выполнения приложения. Значения свойств, определяемые в панели *Properties* среды IDE, устанавливаются раз и навсегда, если только вы не располагаете кодом, который изменяет их впоследствии.

Пока закончим настройку всего элемента управления и займемся настройкой заголовков столбцов.

6.1.4.6 Настройка заголовков столбцов. Если хотите самостоятельно определить внешний вид заголовков столбцов, значение свойства `EnableHeadersVisualStyles` потребуется установить равным `false`:

```
dataGridView->EnableHeadersVisualStyles - false;
```

Обычно элементы управления в приложении `Windows Forms` отображаются в соответствии с действующей темой визуальных стилей, которая и определяет внешний вид элементов управления. При выполнении приложения в среде Windows XP элементы управления отображаются в соответствии с текущей темой Windows XP. Когда значением свойства `EnableHeadersVisualStyles` является `true`, визуальные стили заголовков столбцов будут установлены согласно действующей для приложения теме визуальных стилей, а стили, определенные непосредственно в приложении, будут игнорироваться.

Мы собираемся определить несколько свойств, определяющих внешний вид заголовков столбцов. Простой способ достижения этого – создание объекта `DataGridViewStyle`, для которого можно определить необходимые свойства, и последующая установка этого объекта в качестве определяющего стили заголовков. Объект `DataGridViewStyle` можно создать так:

```
DataGridViewCellStyle^headerStyle=gcnew DataGridVlewCellStyle;
```

Было бы прекрасно, если бы заголовок отображался более крупным шрифтом, и его можно определить, устанавливая значение свойства `Font`:

```
headerStyle->Font=gcnew System::Drawing::Font("Times New
Roman", 12, FontStyle::Bold);
```

Теперь текст заголовка отображается символами шрифта Times New Roman с полужирным начертанием и размером в 12 пунктов.

Для ячеек заголовков можно определить также цвета фона и изображения:

```
headerStyle->BackColor=Color::AliceBlue;
headerstyle->ForeColor=Color::BurlyWood;
```

Текст отображается цветом BurlyWood на фоне AliceBlue. Если вы предпочитаете какие-либо другие цвета, класс Color предоставляет множество возможностей, и средство Intellisense должно отобразить их список по завершении ввода операции разрешения области определения.

Чтобы внешний вид ячеек заголовков соответствовал свойствам, установленным для объекта headerStyle, необходимо добавить следующий оператор:

```
dataGridView->ColuranHeadersDefaultCellStyle=headerStyle;
```

Он устанавливает в качестве значения свойства ColumnHeadersDefaultCellStyle элемента управления дескриптор headerStyle. Тем самым заменяется существующий объект DataGridViewCellStyle, который действовал по отношению к заголовкам.

Применительно к заголовкам столбцов необходимо выполнить еще одну операцию. Более крупный шрифт требует соответствующей подстройки высоты ячеек. Вызов функции AutoResizeColumnHeadersHeight() элемента управления настраивает высоту ячеек заголовков в соответствии с их текущим содержимым:

```
dataGridView->AutoResizeColumnHeadersHeight();
```

В результате высота всех ячеек заголовков будет автоматически подобрана так, чтобы вмещать наибольшее по высоте содержимое. Если требуется автоматически настроить высоту заголовка только конкретного столбца, можно воспользоваться перегруженной версией функции, которая принимает аргумент, указывающий индекс настраиваемого столбца.

Если нужно, чтобы заголовки строки или столбца были невидимыми, этого можно достичь, устанавливая значение свойств RowHeadersVisible и / или ColumnHeadersVisible элемента управления равными ***false***.

6.1.4.7 Форматирование столбца. Первый столбец содержит дескрипторы объекта DateTime. В настоящий момент для получения каких-либо данных для отображения приложение просто вызывает функцию ToString () для объектов, но мы можем его усовершенствовать. Для свойства DefaultCellStyle столбца можно установить свойство Format, и затем эта спецификация формата будет использоваться для отображения содержимого ячеек:

```
dataGridView->Columns[0]->DefaultCellStyle->Format=L"y";
```

Этот оператор устанавливает в качестве значения свойства Format строку, содержащую спецификацию формата y для объекта DateTime, которая представляет его в краткой форме даты в виде месяца и года. Для объектов DateTime существует еще несколько спецификаторов формата, которые можно было бы использовать. Например, спецификатор D отображает день, месяц и год, а спецификаторы f и F, наряду с датой, отображают и время.

Цвет фона ячеек строки определен в свойстве SelectionBackColor свойства DefaultCellStyle. Можно также выбрать отдельную ячейку, щелкая на ней. При этом ее цвет фона изменится на зеленый.

Возможность сортировки строк по любому из столбцов встроена в элемент управления `DataGridView`. Можете щелкнуть на заголовке столбца и убедиться, что строки сортируются по выбранному столбцу. Если вторично щелкнуть на заголовке столбца, порядок сортировки строк изменится на противоположный. К каждому столбцу можно добавить контекстную подсказку, описывающую возможности сортировки.

6.1.4.8 Настройка внешнего вида чередующихся строк. При отображении множества одинаковых по виду строк может оказаться затруднительным фиксировать взгляд на нужной строке. Для облегчения этой задачи можно чередовать цвета строк, устанавливая другой цвет свойства `BackColor` (Цвет фона) для свойства `AlternatingRowsDefaultCellStyle` объекта управления:

```
dataGridView->ALternatingRowsDefaultCellStyle->BackCoLor
=Color::Blue;
```

Вероятно, для обеспечения приемлемого контраста между текстом и фоном потребуется изменить также значение свойства `ForeColor` для свойства `AlternatingRowsDefaultCellStyle`:

```
dataGridView->AlternatingRowsDefaultCellStyle->ForeColor
=Color::White;
```

Теперь чередующиеся строки отображаются поочередно на розовом и синем фоне. Теперь строки легко различать, а белый текст на синем фоне отчетливо виден. По-прежнему строки можно выбирать, щелкая на заголовке строки, при этом выбранная строка выделяется зеленым цветом. Щелчок на ячейке, расположенной слева от заголовков столбцов, приводит к выбору всех строк.

6.1.4.9 Динамическое определение стилей ячеек. Существует несколько возможностей изменения внешнего вида ячеек через обработку событий элемента управления `DataGridView`. Событие `CellFormatting` элемента управления `DataGridView` запускается, когда содержимое ячейки должно быть сформатировано в вид, готовый для отображения, поэтому, добавляя обработчик этого события, внешний вид любой ячейки можно настраивать в соответствии с ее содержимым (табл. 6.4).

Эти свойства позволяют выяснять все необходимые сведения о форматированной ячейке – индекс ее строки и столбца, действующий стиль и содержимое ячейки.

Первый шаг по обработке события `CellFormatting` состоит в определении для него функции обработчика. Код обработчика должен быть максимально кратким и эффективным, поскольку функция вызывается для каждой ячейки элемента управления каждый раз, когда возникает необходимость форматирования ячейки.

Таблица 6.4 – Свойства объекта `DataGridViewCellFormattingEventArgs`

Свойство	Описание
`Value`	Значение этого свойства – дескриптор содержимого ячейки, которая должна быть форматирована
`DeairedType`	Значение этого свойства – дескриптор объекта типа `Type`, который идентифицирует тип содержимого форматируемой ячейки
`CellStyle`	Извлекает или устанавливает стиль ячейки, связанной с событием форматирования, потому это значение – дескриптор объекта типа `DataGridViewCellStyle`
`ColumnIndex`	Значение этого свойства – индекс столбца форматируемой ячейки
`RowIndex`	Это значение – индекс строки форматируемой ячейки
`FormattingApplied`	Значение этого свойства, которое может принимать значение `True` или `False`, указывает, применялось ли форматирование к содержимому ячейки

Элемент управления `DataGridView` определяет события `CellMouseEnter` и `CellMouseLeave`, запускаемые при помещении указателя мыши на ячейку или при перемещении его от нее. Обработчики этих событий можно было бы реализовать так, чтобы ячейка с помещенным на нее указателем мыши выделялась за счет изменения *цвета* фона. Обработчик события `CellMouseEnter` мог бы устанавливать новые цвета изображения и фона, а обработчик события `CellMouseLeave` – восстанавливать первоначальные цвета. Решение этой задачи имеет несколько сложных аспектов, поэтому заслуживает особого рассмотрения.

6.2 Пример выполнения работы

Создать Windows-приложение, которое предлагает пользователю ввести количество строк и столбцов для создания таблицы, в которую будут заноситься элементы матрицы, затем по выбору пользователя определяет: произведение отрицательных элементов матрицы и количество четных элементов матрицы.

Ход выполнения

1 Войти в среду ***Visual Studio 2010*.**

2 В окне **Создать Проект** следует развернуть узел Visual C++, обратиться к пункту `CLR` и на центральной панели выбрать *Приложение* `Windows Form`.

3 Затем в поле редактора **Имя** (где по умолчанию имеется <Введите имя>) следует ввести имя проекта, **Lab6**. В поле **Расположение** можно указать путь размещения проекта или выбрать путь размещения проекта с помощью клавиши (кнопки) **Обзор** (например, N:\\CI\2_trim\Lab6).

4 Для формы изменить значение свойства **Text**, занеся, например, следующие данные: ***«Выполнил студент группы ЭСА-11-1 Иванов П. А. Лабораторная работа 6».***

5 Свойству **FormBorderStyle** формы (стиль рамки окна) присвоить значение ***FixedToolWindow***. Это значение определяет окно как диалоговое, его размеры на этапе прогона приложения (в процессе работы приложения) не могут быть изменены.

6 В верхней части окна разместить контейнер **GroupBox1** и в свойстве **Text** занести текст «***Ввод данных матрицы***».

7 На этом же контейнере **GroupBox1** разместить семь компонентов: **Label1**, **Label2**, **Label3**, **Label4**, **TextBox1**, **TextBox2** и **DataGridWiew1.** Для первого компонента **Label1** в свойство **Text** занести текст «***Задайте размер матрицы***». Для второго компонента **Label2** в свойство **Text** занести текст «***N-строк***». Для третьего компонента **Label3** в свойство **Text** занести текст «***M-столбцов***». Для четвертого компонента **Label4** в свойство **Text** занестти текст «***Матрица A[n,m]***».

8 Выделить компонент **DataGridView1** и для свойств **RowHeadersVisible** (отображение заголовка строк) и **ColumnHeadersVisible** (отображение заголовка столбцов) выбрать параметр ***False***.

9 Ниже контейнера **GroupBox1** разместить контейнер **GroupBox2** и в свойстве **Text** занести текст «***Найти***».

10 Выделить компонент **DataGridView2** и на него разместить два компонента **checkBox1** и **checkBox2.**

11 Выделить первый компонент **checkBox1** и в свойство **Text** занести текст «***произведение отрицательных элементов матрицы***». Аналогично для компонента **checkBox2** в свойство **Text** занести текст «***количество четных элементов матрицы***». Напротив этих компонентов **checkBox** разместить два компонента: **TextBox3** и **TextBox4.**

12 Ниже компонента **GroupBox2** разместить три компонента: **Button1**, **Button2** и **Button3**. На первом компоненте **Button1** в свойство **Text** занести текст «***Создать таблицу***». Для второго компонента **Button2** в свойство **Text** занести текст «***Перенести данные из таблицы в массив и решить задание***». Для третьего компонента **Button3** в свойство **Text** занести текст «***Выход***».

13 В результате проведенных операций должна получиться форма примерно такого вида, как показано на рис. 6.2.

14 Перейти к коду программы и после строки

```
#pragma once
```

подключить библиотеки использования математических функций, описание переменных и размера массива, для этого вставить следующие строки:

```
#include<math.h>
int m, n,kol,kol2,p;
int A[50][50];
```

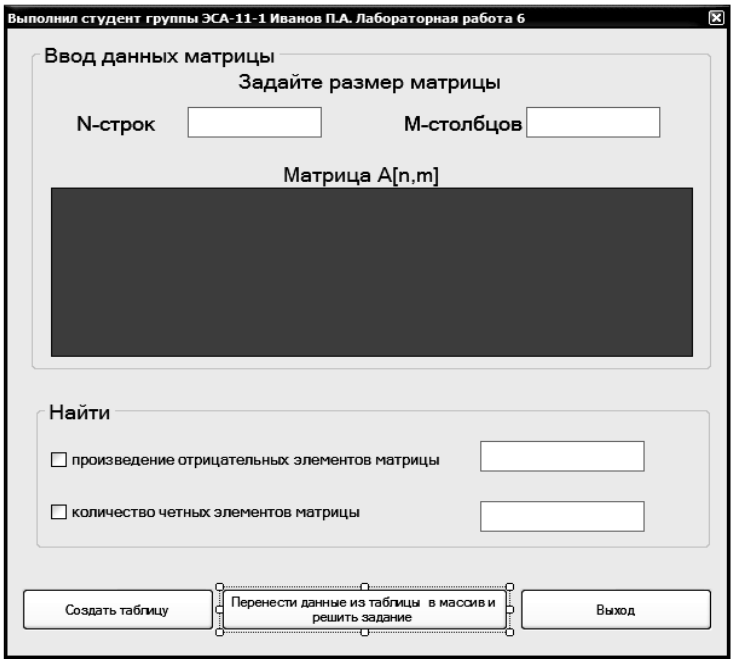


Рисунок 6.2 – Окно формы программы

15 Создать событие ***Click*** для кнопки **Button1** с надписью ***«Создать таблицу»***. Для этого сделать двойной щелчок левой кнопкой мыши по компоненту **Button1**. Внести изменения в код подпрограммы:

```
//Проверка, что не пустые компоненты textBox1 и textBox2
     if ((textBox1->Text!="")&&(textBox2->Text!=""))
     {m = Convert::ToInt32(textBox1->Text);
     n = Convert::ToInt32(textBox2->Text);
//Чистка столбцов компонента DataGridView, если они не пусты
     dataGridView1->Columns->Clear();
//Заполнение компонента DataGridView столбцами
     dataGridView1->ColumnCount = n;
//Заполнение компонента DataGridView строками
     dataGridView1->RowCount = m;}
       else
{MessageBox::Show( "Заполните, пожалуйста, данные", "Ошибка ввода
данных",
MessageBoxButtons::OK, MessageBoxIcon::Exclamation );}
```

16 Создать событие ***Click*** для кнопки **Button2** с надписью ***«Перенести данные из таблицы в массив и решить задание»***. Для этого сделать двойной щелчок левой кнопкой мыши по компоненту **Button2** и в полученную заготовку подпрограммы вставить код для расчетов:

```
//переменную kol и kol2 обнуляем, а переменную p присваиваем единице
kol=0;kol2=0;p=1;
//Производим считывание из ячеек таблицы и вносим данные в массив
for (int i = 0; i < m; i++)
          for (int j = 0; j < n; j++)
{
A[i][j] = Convert::ToSingle(this->dataGridView1->Rows[i]-
>Cells[j]->Value);
```

```
if  (A[i][j]<0) {p=p*A[i][j];kol2++;}
if  (A[i][j]%2==0) {kol++;}
}
```
//Вывод данных нахождения произведения отрицательных элементов матрицы
```
if ((checkBox1->Checked==true)&&(kol2!=0)) {this->textBox3-
>Text=Convert::ToString (p);}
else
if (checkBox1->Checked==true) {this->textBox3-
>Text=Convert::ToString ("нет элементов");}
```
//Вывод данных нахождения количество четных элементов матрицы
```
if ((checkBox2->Checked==true)&&(kol!=0)) {this->textBox4-
>Text=Convert::ToString (kol);}
else
if (checkBox2->Checked==true) {this->textBox4-
>Text=Convert::ToString ("нет элементов");}
return;
```

17 Создать событие ***Click*** для кнопки **Button3** с надписью ***«Выход»***. Для этого сделать двойной щелчок левой кнопкой мыши по компоненту **Button3** и в полученную заготовку подпрограммы вставить код выхода из программы (уже использовался в предыдущих лабораторных работах).

18 На примере лабораторной работы 2 сделать проверку на корректность ввода данных (количество строк и столбцов), используя событие **Leave** в окне свойств (в код программы добавить 2 процедуры, проверяющие правильность ввода).

19 Запустить программу на выполнение, нажав на функциональную кнопку **F5**. Получим следующий вид окна (рис. 6.3).

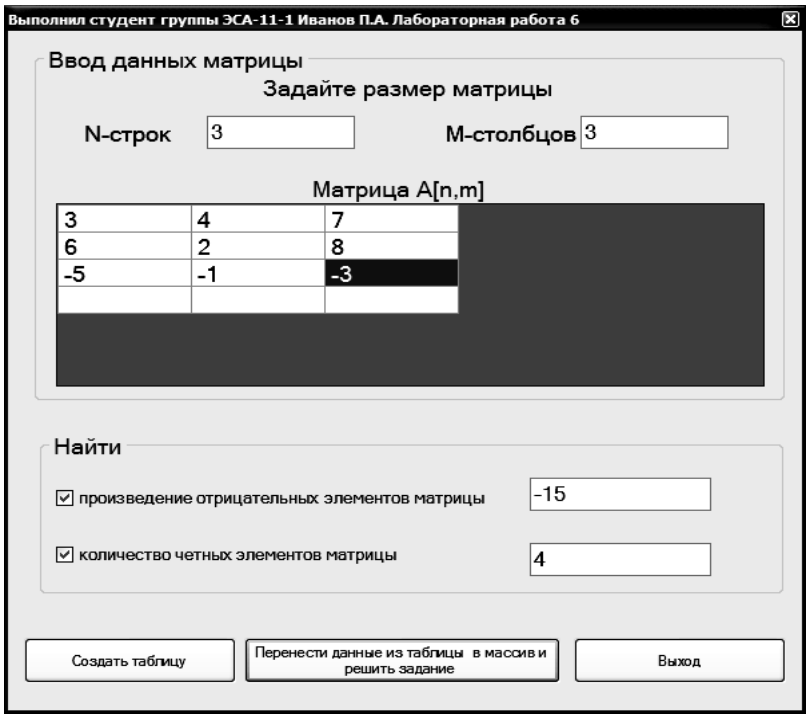


Рисунок 6.3 – Рабочий вид формы приложения

6.3 Рабочее задание

Создать indows-приложение, которое предлагает пользователю задать количество строк и столбцов матрицы, при нажатии на кнопку автоматически создается таблица, в ее ячейки автоматически вводятся элементы матрицы с помощью генератора случайных чисел. Затем по выбору пользователя определяет, соответственно, для каждого варианта задание **а), б)** и **в)**.

Данные взять из таблицы 6.5.

Таблица 6.5 – Индивидуальные задания

Вариант	**Задание**	**Условие задания**
1	2	3
1	***а***	Найти количество элементов, больших заданного числа С (ввод числа С сделать с клавиатуры)
	б	Найти сумму элементов, расположенных по периметру
	в	В матрице А(4;4) найти сумму произведения четных чисел 1-ой строки и произведения положительных чисел 3-го столбца
2	***а***	Найти минимальный по модулю элемент и номер строки и столбца, где он находится
	б	Найти сумму наибольшего положительного и наименьшего четного
	в	Подсчитать количество кратных 3 чисел 2-ой строки и количество четных чисел 1-го столбца матрицы А(6; 6)
3	***а***	Найти произведение элементов, меньших заданного числа Т (ввод числа Т сделать с клавиатуры)
	б	Найти произведение элементов, находящихся на главной диагонали
	в	Найти разность произведения нечетных чисел 3-ей строки и произведения отрицательных чисел 1-го столбца матрицы А(4; 4)
4	***а***	Найти сумму положительных кратных 5 элементов
	б	Найти произведение элементов, находящихся на побочной диагонали
	в	В матрице А(8; 8) найти разность произведения нечетных чисел 3-ей строки и суммы положительных чисел 6-го столбца
5	***а***	Найти произведение отрицательных четных элементов.
	б	Найти сумму элементов, находящихся на главной диагонали.
	в	В матрице А(5; 5) найти сумму количества четных чисел 2-ой строки и количества отрицательных чисел 4-го столбца
6	***а***	Найти количество положительных нечетных элементов
	б	Найти сумму элементов, находящихся на побочной диагонали
	в	В матрице А(3;3) найти произведение количества нечетных чисел 1-ой строки и количества положительных чисел 3-го столбца
7	***а***	Найти количество элементов, меньших числа 5
	б	Найти количество отрицательных элементов, находящихся на главной диагонали
	в	Найти максимальный элемент 3-го столбца и сумму нечетных элементов 1-ой строки матрицы А(5; 5)

Продолжение таблицы 6.5

1	2	3
8	***а***	Найти произведение положительных кратных 3 элементов
	б	Найти количество отрицательных элементов, находящихся на побочной диагонали
	в	В матрице A(6; 6) найти произведение суммы четных чисел в 3-ей строке и суммы отрицательных чисел 1-го столбца
9	***а***	Найти сумму отрицательных некратных 5 элементов
	б	Найти количество четных элементов, находящихся на главной диагонали
	в	Найти произведение суммы положительных чисел в 4-ом столбце на количество четных чисел 2-ой строки матрицы A(6; 6)
10	***а***	Найти квадрат максимального элемента и номер строки и столбца, где он находится
	б	Найти количество четных элементов, находящихся на побочной диагонали
	в	В матрице A(7; 7) найти разность количества положительных чисел 1-ой строки и количества четных чисел 3-го столбца
11	***а***	Найти сумму четных элементов из интервала [–10; 10] матрицы A(4; 4)
	б	Найти количество нечетных элементов, находящихся на главной диагонали
	в	Найти произведение количества четных элементов 3 строки на сумму нечетных элементов 2 столбца матрицы A(4; 4)
12	***а***	Найти количество кратных 3 элементов из интервала [–6; 8] матрицы A(5; 5)
	б	Найти количество нечетных элементов, находящихся на побочной диагонали
	в	В матрице A(5; 5) найти произведение количества четных чисел 2-ой строки и количества отрицательных чисел 4-го столбца
13	***а***	Найти произведение отрицательных нечетных элементов матрицы A(5; 5)
	б	Найти сумму четных элементов, находящихся на главной диагонали
	в	В матрице A(5; 5) найти произведение количества нечетных чисел 3-го столбца и количества отрицательных чисел 3 строки
14	***а***	Найти количество положительных элементов из интервала [–5; 6] матрицы A(6; 6)
	б	Найти сумму четных элементов, находящихся на побочной диагонали
	в	В матрице A(6; 6) найти произведение суммы кратных 3 чисел 2-ей строки и суммы отрицательных чисел 2-го столбца
15	***а***	Найти максимальный по модулю элемент и номер строки и столбца, где он находится
	б	Найти сумму элементов, кратных 3, находящихся на главной диагонали
	в	Найти произведение количества четных чисел во 2-ом столбце на количество нечетных чисел 2-ой строки матрицы A(4; 4)

Продолжение таблицы 6.5

1	2	3
16	***а***	Найти сумму положительных кратных 3 элементов
	б	Найти сумму элементов, кратных 3, находящихся на побочной диагонали
	в	Найти среднее геометрическое нечетных элементов 2-го столбца и количество кратных 5 элементов 3-ей строки матрицы A(5;5)
17	***а***	Найти количество отрицательных четных элементов
	б	Найти сумму отрицательных четных элементов, находящихся на главной диагонали
	в	Найти сумму нечетных элементов 3-го столбца и произведение отрицательных кратных 3 элементов 2-ой строки матрицы A(6; 6)
18	***а***	Найти произведение положительных четных элементов
	б	Найти сумму отрицательных четных элементов, находящихся на побочной диагонали
	в	Найти произведение суммы кратных 3 чисел в 4-ом столбце на количество нечетных чисел 2-ой строки матрицы A(4; 4)
19	***а***	Найти сумму отрицательных четных элементов
	б	Найти произведение элементов, не кратных 3, которые находятся на главной диагонали
	в	В матрице A(7; 7) найти разность количества нечетных чисел 1-ой строки и количества четных чисел 4-го столбца
20	***а***	Найти максимальный элемент, номер строки и столбца, в котором он находится
	б	Найти разницу суммы четных и количества отрицательных элементов матрицы, находящихся на главной диагонали
	в	Найти сумму нечетных элементов 2-го столбца и произведение отрицательных кратных 3 элементов 4-ой строки матрицы A(4; 4)
21	***а***	Найти минимальный элемент, номер строки и столбца, в котором он находится
	б	Найти произведение суммы четных на количество положительных элементов матрицы, находящихся на главной диагонали
	в	Найти произведение отрицательных четных элементов 2-ой строки и количество некратных 5 элементов 2-го столбца матрицы A(5; 5)
22	***а***	Найти количество положительных кратных 5 элементов
	б	Найти произведение суммы четных на количество положительных элементов матрицы, находящихся на побочной диагонали
	в	Подсчитать количество положительных кратных 3 элементов 1-ой строки и количество нечетных элементов 2-го столбца матрицы A(6; 6)
23	***а***	Найти произведение отрицательных нечетных элементов
	б	Найти модуль суммы элементов кратных 5, находящихся на побочной диагонали
	в	Найти разность произведения нечетных чисел 3-ей строки и произведения отрицательных чисел 1-го столбца матрицы A(4; 4)

Продолжение таблицы 6.5

1	2	3
24	***а***	Найти квадрат минимального элемента и номер строки и столбца, где он находится
	б	Найти разницу суммы четных и количества отрицательных элементов матрицы, находящихся на побочной диагонали
	в	В матрице A(7; 7) найти разность произведения нечетных чисел 2-ой строки и суммы положительных чисел 6-го столбца
25	***а***	Найти произведение положительных некратных 5 элементов
	б	Найти модуль суммы элементов, кратных 5, находящихся на главной диагонали
	в	В матрице A(5; 5) найти сумму количества четных чисел 3-ей строки и количества отрицательных чисел 4-го столбца
26	***а***	Найти максимальный по модулю элемент и номер строки и столбца, где он находится
	б	Найти разность максимального и минимального элементов, находящихся на главной диагонали
	в	В матрице A(6; 6) найти произведение количества нечетных чисел 2-го столбца и количества положительных чисел 3 строки
27	***а***	Найти количество отрицательных некратных 3 элементов
	б	Найти разность максимального и минимального элементов, находящихся на побочной диагонали
	в	Найти максимальный элемент 2-ой строки и количество четных элементов 5-го столбца матрицы A(5; 5)
28	***а***	Найти произведение положительных нечетных элементов
	б	Найти модуль разности суммы четных и произведения нечетных элементов матрицы, находящихся на главной диагонали
	в	В матрице A(6; 6) найти произведение суммы четных чисел 3-ей строки и суммы отрицательных чисел 1-го столбца
29	***а***	Найти сумму отрицательных нечетных элементов
	б	Найти модуль разности суммы четных и произведения нечетных элементов матрицы, находящихся на побочной диагонали
	в	Найти произведение суммы положительных чисел 1-й строки на сумму четных чисел 2-го столбца матрицы A(5; 5)
30	***а***	Найти произведение отрицательных четных элементов
	б	Найти произведение максимального элемента главной диагонали на минимальный элемент побочной диагонали
	в	В матрице A(7; 7) найти разность количества отрицательных чисел 2-ой строки и количества нечетных чисел 3-го столбца

6.4 Контрольные вопросы

1 Понятие матрицы в языке **C++**.

2 Как организуется индексирование числовых массивов в языке **C++**?

3 В какой очередности и как происходит заполнение матрицы на языке **C++**?

4 Условия селективной обработки элементов матрицы.

5 Условия обработки элементов строк, столбцов и диагоналей матрицы.

6 С помощью каких визуальных компонентов можно сделать ввод и вывод элементов матрицы в ручном режиме?

7 Свойства компонента `DataGridView`.

8 Сколько потребуется операторов цикла (каких) для вывода элементов матрицы в компонент `DataGridView`?

7 ЛАБОРАТОРНАЯ РАБОТА 7. ИЗУЧЕНИЕ ВЕРОЯТНОСТНЫХ АЛГОРИТМОВ

Цель: Использование накопленных знаний для решения практической задачи: применение метода Монте-Карло для определения площади плоской фигуры.

7.1 Теоретические сведения

Метод Монте-Карло (статистических испытаний) – это один из методов статистического моделирования, основанный на кибернетической идее «черного ящика». «Черный ящик» означает такую систему, которая изучается с помощью сопоставления воздействий на ее входы и реакций на ее выходы без анализа происходящих в ней процессов. Он применяется в тех случаях, когда построение аналитической модели является затруднительным или невозможным.

Смысл метода Монте-Карло состоит в том, что исследуемый процесс моделируется путем многократных повторений его случайных реализаций. Единичные реализации называются статистическими испытаниями.

Метод Монте-Карло часто используется для решения физических и математических задач (например, для вычисления двойных и тройных интегралов) с использованием вероятностных механизмов. В рамках этого метода строится вероятностная модель, соответствующая математической или физической задаче, на ней реализуется случайная выборка. Для организации случайной выборки используются специальные подпрограммы – генераторы псевдослучайных чисел. Чем больше выборок, тем точнее получаемый результат.

В примере производится вычисление площади плоской фигуры, ограниченной квадратом, вершины которого представлены точками (100, 100); (100, –100); (–100, –100); (–100,100) (см. рис. 7.1).

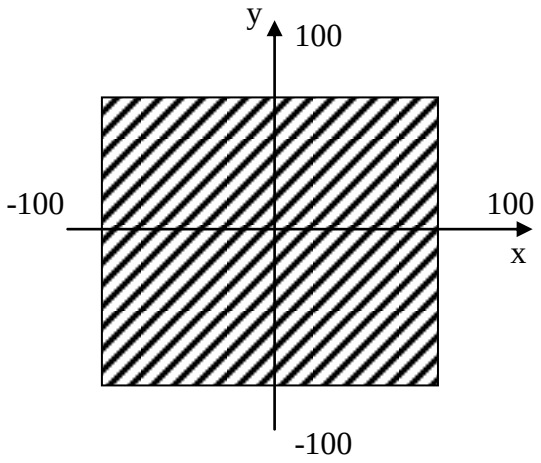


Рисунок 7.1 – Пример фигуры для нахождения площади

Очевидно, что площадь такого квадрата равна 40000. В область квадрата случайным образом помещаются N точек, из которых n попадут на плоскую фигуру, а m = N – n – будут извне. Тогда площадь фигуры легко вычисляется по формуле $S_{фиг} = \frac{n}{N} S_{кв}$, где $S_{кв} = 40000$ – площадь квадрата, внутри которого размещается фигура.

Практически метод Монте-Карло в данном случае реализуется следующим образом: в цикле с помощью генератора псевдослучайных чисел генерируется два случайных числа в пределах от –100 до +100, которые трактуются как координаты точки на плоскости в пределах квадрата. Подсчитывается количество n точек, координаты которых удовлетворяют условию попадания в фигуру.

Для определения ошибки вычисления рассчитывается реальная площадь фигуры Sp. Относительная ошибка (в процентах) определяется по формуле:

$$\Delta\% = \frac{\left|S_p - S_{фиг}\right|}{S_p} 100\% .$$

Естественно, такое определение погрешности делается только для демонстрации точности метода Монте-Карло. При реальном его применении реальную площадь фигуры Sp найти нельзя, так как в противном случае незачем использовать метод Монте-Карло.

7.2 Пример выполнения работы

Методом Монте-Карло определить площадь фигуры, изображенной на рис.7.2. Вычислить фактическую площадь и сравнить ее с расчетной. Определить относительную погрешность. Определить составные части границ фигуры.

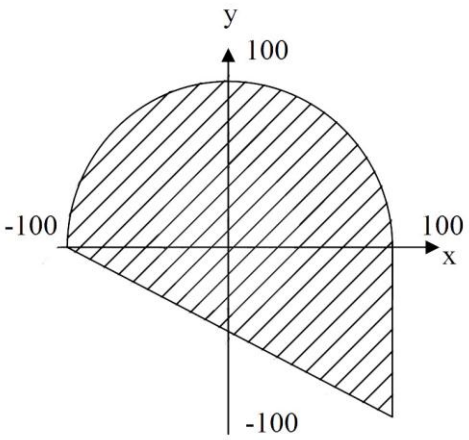


Рисунок 7.2 – Пример фигуры для нахождения площади методом Монте-Карло

Ход выполнения

1 Определить составные части границ фигуры: из фигуры на рисунке можно сделать вывод, что ее граница состоит из трех линий:

– полуокружности (радиус 100) $y = \sqrt{10000 - x^2}$ $-100 \le x \le 100$;

– отрезка прямой $y = -0{,}5(x + 100)$, $-100 \le x \le 100$;

– вертикального отрезка $x = 100$ $-100 \le y \le 0$.

2 Определить расчетную площадь фигуры. Из рисунка видно, что она состоит из двух фигур: полукруга радиусом 100 и прямоугольного треугольника с катетами 100 и 200. Значит, общая площадь фигуры равна: **Sp = 10000·π/2 + 10000**.

3 Войти в среду ***Visual Studio 2010***.

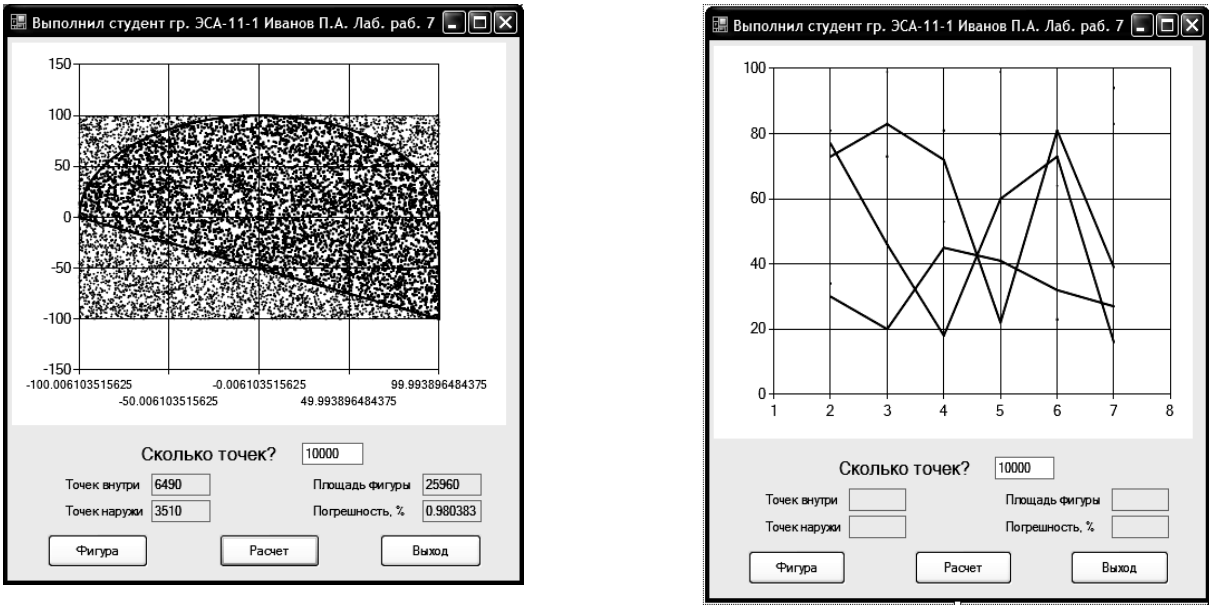
4 В окне **Создать Проект** следует развернуть узел Visual C++, обратиться к пункту `CLR` и на центральной панели выбрать *`Приложение Windows Form`*.

5 Затем в поле редактора **Имя** (где по умолчанию имеется <Введите имя>) следует ввести имя проекта, **`Visual_Lab7`**. В поле **Расположение** можно указать путь размещения проекта или выбрать путь размещения проекта с помощью клавиши (кнопки) **Обзор** (например, N:\\CI\2_trim\Lab7).

6 Для формы изменить значение свойства **`Text`**,занеся, например, следующие данные: ***«Выполнил студент группы ЭСА-11-1 Иванов П. А. Лабораторная работа 7».***

7 Поместить на форму компонент **`Chart1`**. С помощью свойства **`Anchor`** привязать компонент ко всем сторонам формы. Компонент **`Chart1`** должен занимать почти всю область формы и при изменении размеров формы будут, соответственно, меняться и его размеры. Это не лучший способ привязки размеров графика к размерам формы: возможны искажения пропорций фигуры по высоте и ширине. Лучше делать перерасчет размеров компонента **`Chart1`** в обработчике события **`Resize`** (изменение размеров) для формы, назначая одинаковый размер как по высоте, так и по ширине, определяя для этого минимальный из размеров доступной клиентской области по высоте или ширине (заранее определить, какую по высоте область занимают остальные компоненты, размещаемые на форме).

8 Разместить на форме ряд компонентов (5 компонентов **`TextBox`**, 5 компонентов **`Label`**, 3 компонента **`Button`** и один компонент **`Chart`**) для отображения результатов работы (например, как показано на рис.7.3). При этом, используя свойство **`Anchor`** (привязка к родительскому компоненту), привязать компоненты, расположенные слева к левому краю формы, а расположенные справа – к правому. Кнопки «***Фигура***», «***Расчет***» и «***Выход***» нужно привязать к нижнему краю. При изменении размеров формы лучше делать перерасчет ширины кнопок, тогда они будут пропорционально увеличиваться, оставаясь «привязанными» к нижнему краю формы. Форма будет иметь вид примерно как на рис. 7.3.



а) *б)*

а – на этапе проектирования; б – после выполнения расчета

Рисунок 7.3 – Общий вид исходной формы

9 Для компонентов `TextBox2`, `TextBox3`, `TextBox4` и `TextBox5` в свойстве `ReadOnly` (только чтение) присвоить значение *`true`*, запрещающее пользователю заносить в компонент какие-либо данные. Для компонента `TextBox1` данные будут вводиться с клавиатуры с помощью компонента `TextBox`. В свойства `Text` этого компонента ввести значение «10000» – значение по умолчанию. Это значение будет показываться при запуске приложения на выполнение. При выполнении приложения его можно будет заменить другим.

10 Через свойство **`Series`** компонента **`Chart1`** вызвать окно **`Редактора коллекции Series`**. Добавить **`Series`** для отображения границ фигуры (по количеству составных частей контура фигуры). В исходном примере – три графика. Тип графиков выбрать **`Line`** (линия) или **`Spline`** (полилиния). Цвет графиков взять одинаковый. Добавить еще два **`Series`** типа **`Point`** (точки) для демонстрации случайных точек (всего должно быть для данного примера 5 **`Series`**). Выбрать по своему усмотрению цвет этих точек, размер **Маркера** определить равным 1.

11 В объектном коде создаваемого проекта добавить присоединение библиотек:

```
#pragma once
#include <cmath>
#include <stdlib.h>
#include <time.h>
#define _USE_MATH_DEFINES
#include <math.h>
```

12 В разделе ***using*** добавить строку:

```
using namespace System::Windows::Forms::DataVisualization::Charting;
```

13 Сделать двойной щелчок левой кнопкой мыши по кнопке «***Фигура***», при этом в компоненте **Chart1** рисуется контур фигуры. Каждая составная часть (их три) рисуется в своей серии. При отображении они соприкасаются друг с другом, образуя единый контур. Вставить в заготовку кода следующий код программы:

```
Series^ plot1 = chart1->Series[0];
Series^ plot2 = chart1->Series[1];
Series^ plot3 = chart1->Series[2];
double y, x;
// Рисуем верхнюю часть фигуры – полуокружность радиуса 100
for (int i=-100; i<=100; i++)
{ x=i;
y=sqrt(10000-x*x);
plot1->Points->AddXY(x, y);
}
// Рисуем вторую часть - наклонную прямую y=-0,5(x+100)
for (int i=-100; i<=100; i++)
{ x=i;
y=-0.5*(x+100);
plot2->Points->AddXY(x, y);
}
// Рисуем вертикальную прямую x=100.
for (int i=-100; i<=0; i++)
{x=100;
y=i;
plot3->Points->AddXY(x, y);
}
```

14 Весь расчет выполняется в обработчике щелчка на кнопке «**Расчет»**. С помощью генератора псевдослучайных чисел **rand** генерируются пары псевдослучайных чисел, которые трактуются как координаты случайной точки внутри квадрата. Точки отображаются на экране в сериях типа ***Point*** (точки) компонента **Chart1**. Одновременно идет подсчет количества точек, попавших в область и не попавших. По завершении процесса генерации случайных точек осуществляется расчет итогов. Для этого сделать двойной щелчок левой кнопкой мыши по кнопке «***Расчет***» и в полученную заготовку кода вставить следующий код:

```
Series^ plot4 = chart1->Series[3];
Series^ plot5 = chart1->Series[4];
int T_vn, T_za, n;
float x, y, S_fig, S_r,er;
// Очищаем графики (точки) для проведения повторного Расчета
plot4->Points->Clear();
plot5->Points->Clear();
if (textBox1->Text!="")
          {
// Запрашиваем количество точек
n = Convert::ToInt32(textBox1->Text);
// Очищаем счетчики точек
T_vn = 0; T_za = 0;
```

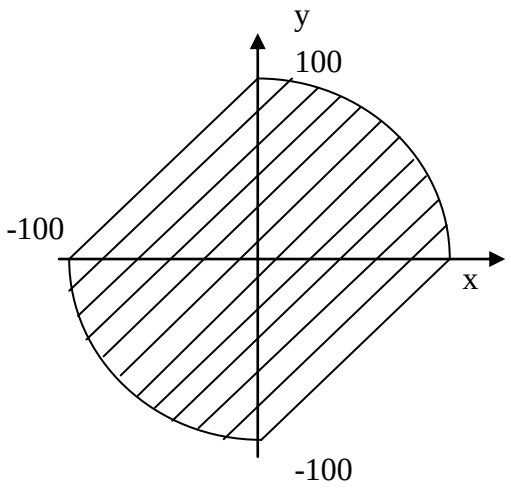
```
for (int i=1; i<=n; i++)
{
// Генерируем пару случайных чисел и трактуем их как
// координаты случайной точки
x = rand()%201-100;
y = rand()%201-100;
// Проверяем, куда попадает точка: внутрь фигуры или вне ее.
// В зависимости от этого выбираем цвет точки.
// Кроме того, ведем подсчет отдельно внешних и внутренних точек
if
(((x*x+y*y<=10000)&&(y>=0))||((y<0)&&(x<=100)&&(y>=-0.5*(x+100))))
{ plot4->Points->AddXY(x, y); T_vn++;}
else
{ plot5->Points->AddXY(x, y); T_za++;}
}
// Количество точек внутри фигуры
textBox2->Text = Convert::ToString (T_vn);
// Количество точек вне фигуры
textBox3->Text = Convert::ToString (T_za);
// Площадь фигуры, определенная по методу Монте-Карло
S_fig = 40000*T_vn/n;
//Площадь реальная
S_r= 10000+M_PI*10000/2;
// Ошибка
er = fabs(S_r-S_fig)/S_r*100;
textBox4->Text = Convert::ToString (S_fig);
textBox5->Text = Convert::ToString (er);
}
else MessageBox::Show( "Введите количество точек", "Ошибка
ввода данных",
MessageBoxButtons::OK, MessageBoxIcon::Exclamation );
}
```

15 Аналогично для третьей кнопки «Выход» сделать двойной щелчок левой кнопкой мыши и вставить в заготовку код, который будет производить выход из программы (уже неоднократно использовался в предыдущих лабораторных работах).

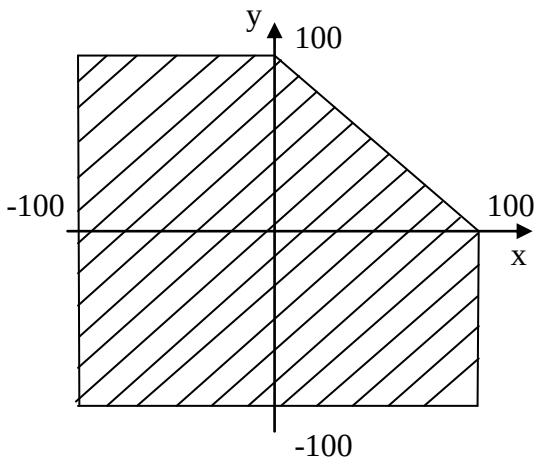
7.3 Рабочее задание

Составить программу для определения методом Монте-Карло площади заштрихованой фигуры. Вид фигуры, в зависимости от варианта, выбрать из таблицы 7.1 (выбор плоской фигуры выполняется следующим образом: вариант 1 – а, вариант 2 – б, …, вариант 12 – м; вариант 13 – а, вариант 14 – б, …вариант 24 – м; вариант 25 – а; вариант 26 – б, …вариант 36 – м). Вычислить фактическую площадь и сравнить ее с расчетной.

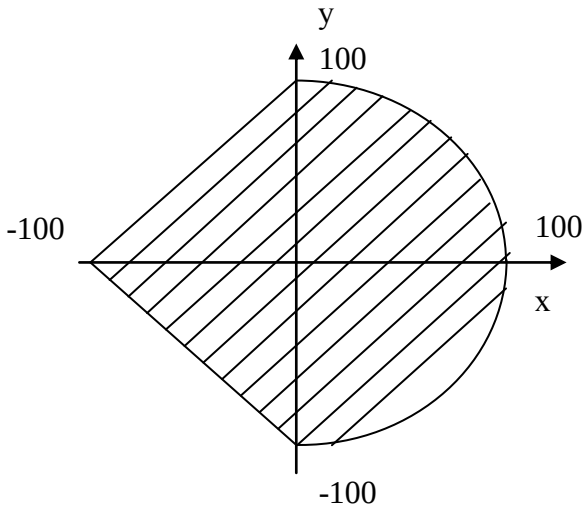
Таблица 7.1 – Виды плоских фигур



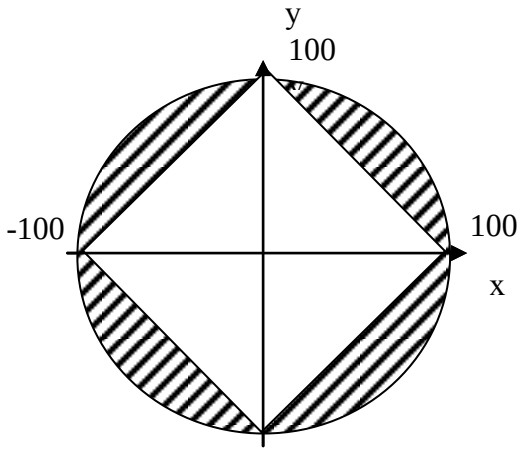
а)



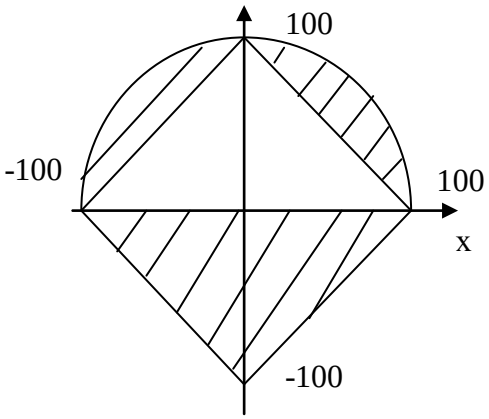
б)



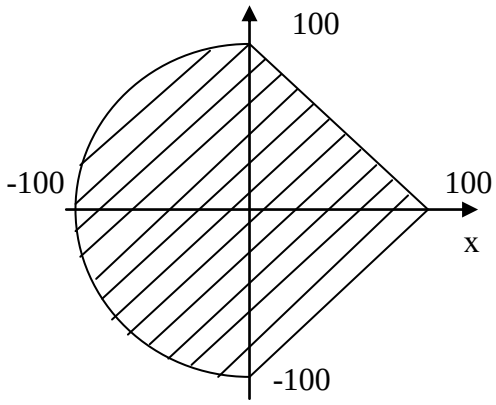
в)



г)

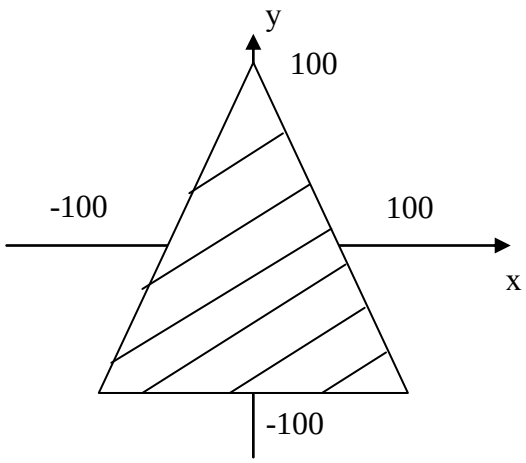


д)

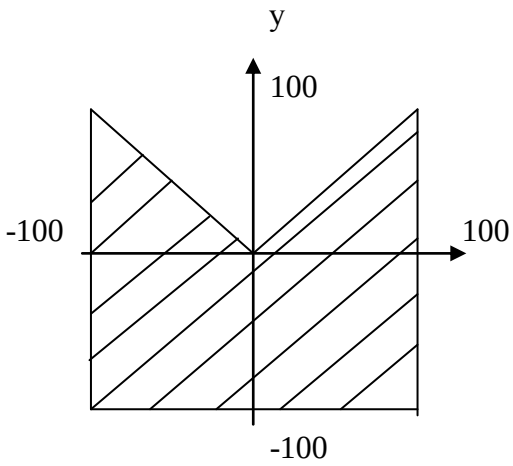


е)

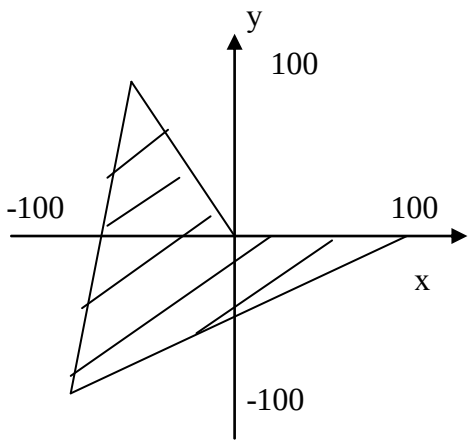
Продолжение таблицы 7.1



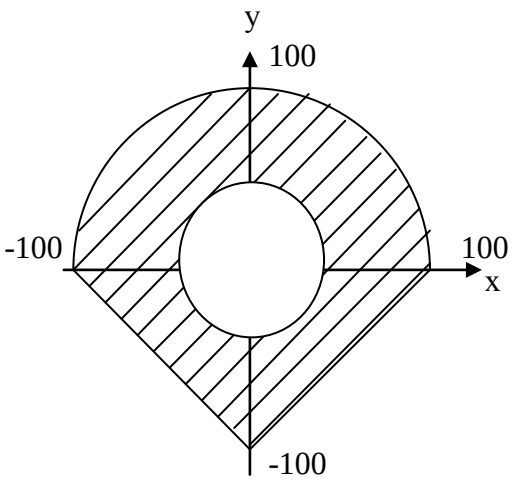
ж)



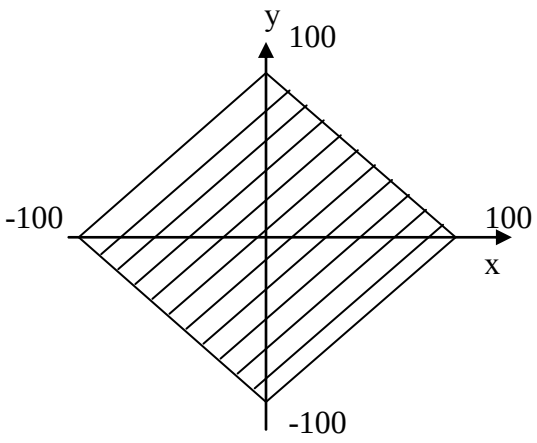
з)



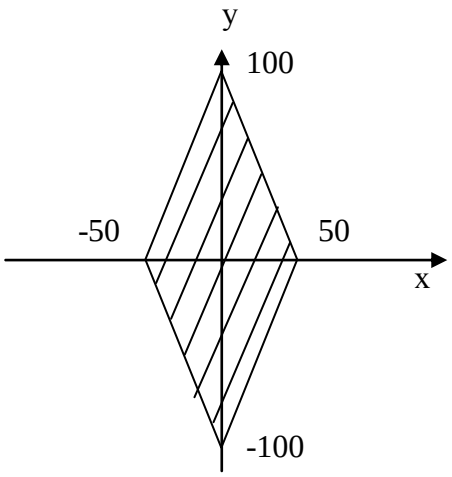
и)



к)



л)



м)

8 ЛАБОРАТОРНАЯ РАБОТА 8. РАБОТА С ДИАЛОГОВЫМИ ОКНАМИ. СОЗДАНИЕ ОПЕРАЦИОННОГО МЕНЮ

Цель: научиться создавать операционное меню формы, использовать диалоговые окна: `FontDialog`, `ColorDialog` и `SaveDialog` для работы с данными и передачи их в текстовые файлы.

8.1 Теоретические сведения

8.1.1 Понятие операционного меню

Процесс индивидуальной настройки графического интерфейса пользователя в соответствии с потребностями конкретного приложения предполагает добавление стандартных элементов управления и дополнительных форм, а также настройку их свойств.

Чтобы добавить элемент управления `MenuStrip` (*операционное меню*) в верхнюю часть формы, сначала нужно его выбрать в окне `Панели элементов`, а затем сделать щелчок по форме, куда его нужно поместить, в верхней части клиентской области окна приложения. Слева над элементом управления отобразится небольшая стрелка. Щелчок на ней приведет к открытию всплывающего окна, показанного на рис. 8.1.

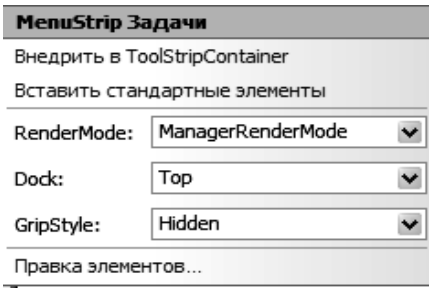


Рисунок 8.1 – Окно MenuStrip задачи

Первый элемент во всплывающем окне `MenuStrip Задачи` (Задачи строки меню) внедряет строку меню внутрь элемента управления `ToolStrip Container Задачи`, предоставляющего панели, расположенные вдоль всех четырех сторон формы, и центральную область, в которой можно помещать другой элемент управления (рис. 8.2).

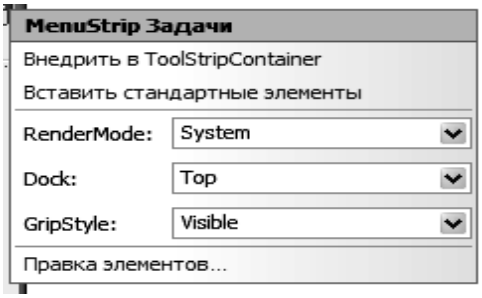


Рисунок 8.2 – Окно `ToolStrip Container Задачи`

Второй элемент «*Внедрить стандартные элементы*» генерирует четыре стандартных элемента меню: *Файл*, *Правка*, *Сервис* и *Справка*, содержащие списки соответствующих пунктов меню. В то же время, добавление этого меню предоставляет то преимущество, что меню *Файл* будет содержать пункт *Выход*, позволяющий закрывать приложение. Элемент `RenderMode` позволяет выбирать стиль изображения строки меню, и для него можно оставить значение, выбранное по умолчанию. Опция `Dock` (*Стыковка*) позволяет выбирать сторону формы, к которой будет пристыкована строка меню, или же оставлять строку меню в отстыкованном состоянии. Опция `GripStyle` определяет видимость рамки для перетаскивания меню. В *Visual Studio 2010* строки меню снабжены видимыми рамками для перемещения. Выбор последней опции `Правка элементов` приводит к отображению диалогового окна, в котором можно изменять свойства строки меню или любых его элементов.

Чтобы добавить меню, нужно ввести текст меню в поле строки меню. Например, &*Сохранить данные таблицы в файл*. Символ & предшествует символу, выполняющему роль клавиши быстрого доступа для данного элемента меню, так например, клавиатурная комбинация <*Ctrl+S*> служит комбинацией быстрого доступа к меню *Сохранить данные таблицы в файл* (рис. 8.3).

Рисунок 8.3 – Окно в процессе разработки меню

Меню *Файл* будет содержать два подменю: *Сохранить данные таблицы файл* и *Выход*, однако щелчок по каждому из них будет генерировать событие. Доступ к свойствам меню можно получить, щелкая правой кнопкой мыши на элементе управления в форме (например, пункт *Меню*) и выбирая из контекстного меню пункт `Правка DropDownItems`. В результате откроется диалоговое окно *Редактор коллекции элементов*, показанное на рис. 8.4.

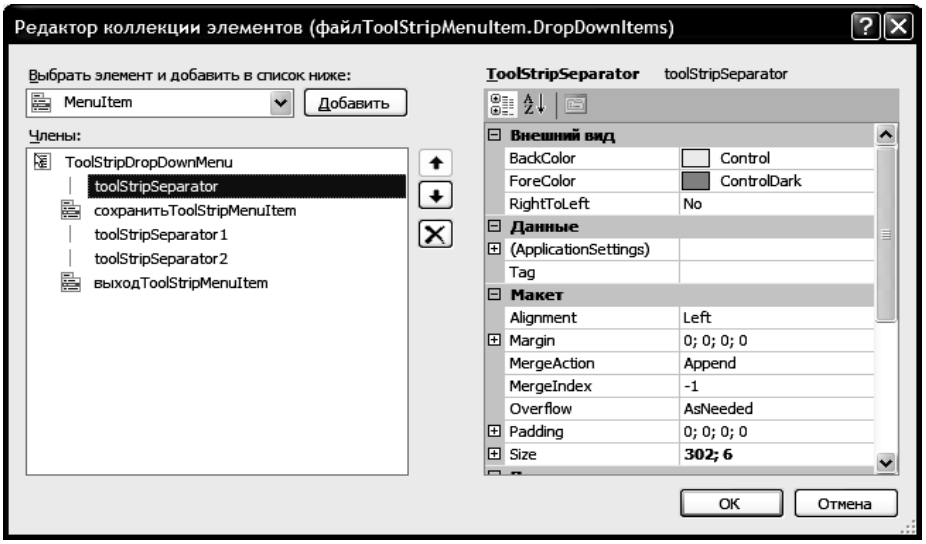


Рисунок 8.4 – Окно редактора коллекции элементов

Выбор в левой панели диалогового окна имени, соответствующего меню *Файл*, приведет к отображению его свойств на правой панели. После этого можно изменить значение свойства (Name). На рис. 8.4 представлено диалоговое окно после внесения изменений. Преимущество использования этого диалогового окна состоит в том, что оно позволяет работать со всеми элементами строки меню, настраивая их свойства должным образом. Кроме того, окно позволяет добавлять новые элементы в строку меню или изменять порядок отображения пунктов.

Новые элементы меню можно добавлять, используя диалоговое окно *Редактор коллекции элементов*, показанное на рис. 8.4. Для этого достаточно выбрать меню (или строку меню), которое требуется добавить, и щелкнуть на кнопке *Добавить*. Кроме того, с помощью панели *Редактор* можно работать непосредственно со строкой меню.

Чтобы добавить элементы в меню, нужно щелкнуть на нем в строке меню. Затем щелкнуть на расположенном под ним элементе меню и ввести нужный текст. Для этого элемента можно также добавить клавишу быстрого доступа. Для этого потребуется щелкнуть на стрелке вниз в колонке значения свойства `ShortcutKeys`, в результате чего откроется список, показанный на рис. 8.5.

Выбрать клавишу или клавиши-модификаторы, установив соответствующие флажки, и выбрать клавишу быстрого доступа из раскрывающегося списка. Значения свойства `ToolTipText` позволяет ввести текст, который является подсказкой при наведении курсора мышки на пункт меню. Установка значения свойства `AutoToolTip`, расположенного в начальной части списка, равным `True`, приведет к отображению подсказки, текст которой будет совпадать с текстом элемента меню. Если же оставить это значение равным `False`, в качестве текста подсказки будет использовано значение свойства `ToolTipText`. Устанавливая значение свойства `ShowShortcutKeys`, можно также управлять отображением комбинации клавиш быстрого доступа вместе с текстом элемента меню.

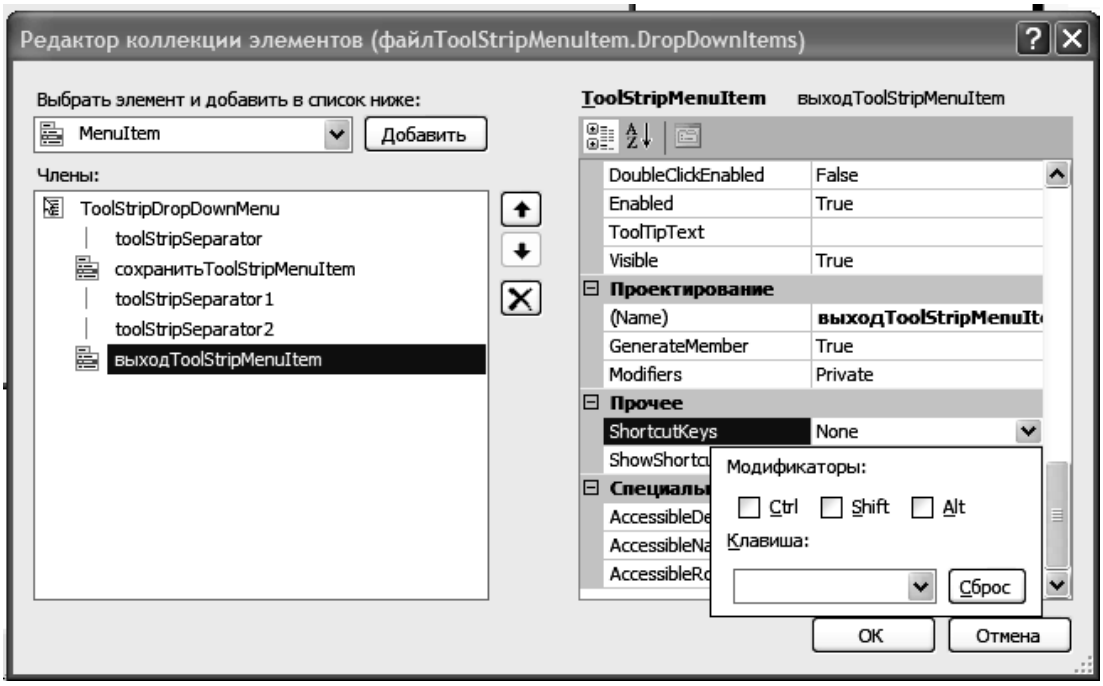


Рисунок 8.5 – Окно для изменения комбинации клавиш быстрого доступа

8.1.2 Обработчик событий для элементов операционного меню

Начнем с создания функции обработчика события для элемента операционного меню Файл. Двойной щелчок на этом элементе меню создает следующий код обработчика события:

```
private: System::Void ФайлMenuItem_Click(System::Object^ sender.
System::EventArgs^ e){}
```

В этом скелете обработчика тело функции не содержит никакого кода. Первый параметр – дескриптор, ссылающийся на элемент управления, выдавший событие, а второй аргумент предоставляет информацию о самом событии. Тип первого аргумента соответствует типу элемента управления, инициировавшего событие – в данном случае это `ToolStripMenuItem`, поскольку функция обработчика вызывается при щелчке на элементе *меню Файл*. Дескриптор элемента меню хранится в члене `ФайлMenuItem` класса `Forml` и его тип можно проверить в определении класса. Аналогично, фактический тип второго аргумента обработчика события `Click` также зависит от типа элемента управления.

Имя функции обработчика, генерируемое по умолчанию, – `ФайлMenuItem_Click`. Не следует изменять автоматически сгенерированные имена в коде. Это всегда следует делать в окне `Свойства`. Если имя, созданное для функции обработчика события, вас не устраивает, его можно изменить через свойство события `Click` в окне `Свойства` данного элемента управления.

8.1.3 Элемент управления с вкладками (`TabControl`*)*

Элемент управления `TabControl` предоставляет несколько вкладок, каждая из которых может содержать собственный набор элементов управления. Открыть окно *Панель элементов* (<Ctrl+Alt+X>) и выберать в списке элемент `TabControl` – он расположен в группе *Контейнеры*. Щелкнуть в клиентской области формы и добавьте элемент управления с вкладками, после чего открыть его окно свойств. Для этого нажать клавишу <F4>.

Все элементы управления, перечисленные в группе *Контейнеры*, могут содержать другие элементы управления, и, следовательно, все они предлагают средства объединения элементов управления в группу. Понятно, что каждая вкладка может содержать собственный набор элементов управления, причем элемент управления вкладки может содержать любое количество вкладок.

Нам необходимо, чтобы элемент управления вкладки заполнял всю клиентскую область окна. Это определяется значением свойства `Dock`, расположенного в группе свойств `Макет` элемента управления `TabControl`. Окно `Свойства` элемента управления с вкладками с раскрытой ячейкой значения свойства `Dock` приведено на рис. 8.6.

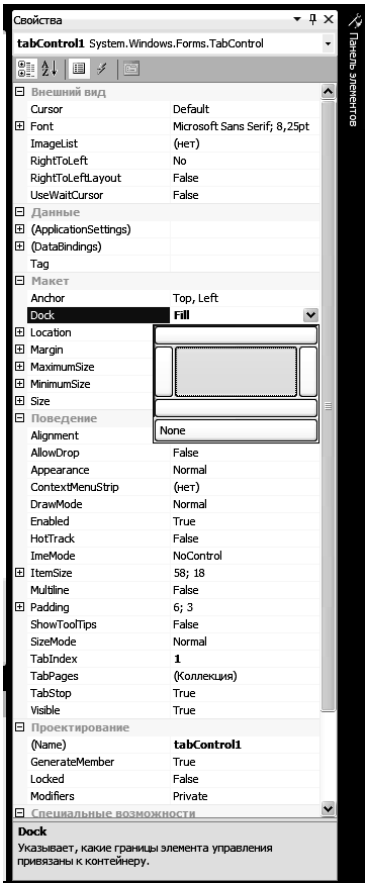


Рисунок 8.6 – Свойства `Dock` *в окне Свойства элемента управления с вкладками*

Элемент управления с вкладками содержит две вкладки, но при необходимости в него можно добавлять новые вкладки, щелкая на кнопке со стрелкой в правом верхнем углу элемента управления и выбирая команду *Добавить вкладку* из всплывающего меню (рис. 8.7).

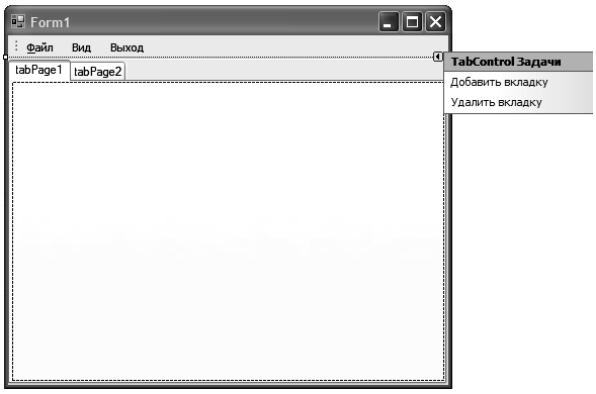


Рисунок 8.7 – Добавление новой вкладки

Необходимо изменить текст каждой вкладки на что-либо более вразумительное. Это же относится к значениям свойства (*Name*). Перейти в окно *Свойства* элемента управления `TabControl` – нажатие клавиши <F4> при выбранном элементе управления приведет к отображению его окна свойств, если оно еще скрыто. Затем выберать поле значения свойства TabPage и щелкнуть на появившейся кнопке с символом многоточия. Откроется диалоговое окно *`Редактор коллекции TabPage`*, показанное на рис. 8.8. Значение свойства `Text` позволяет изменять надпись на вкладке.

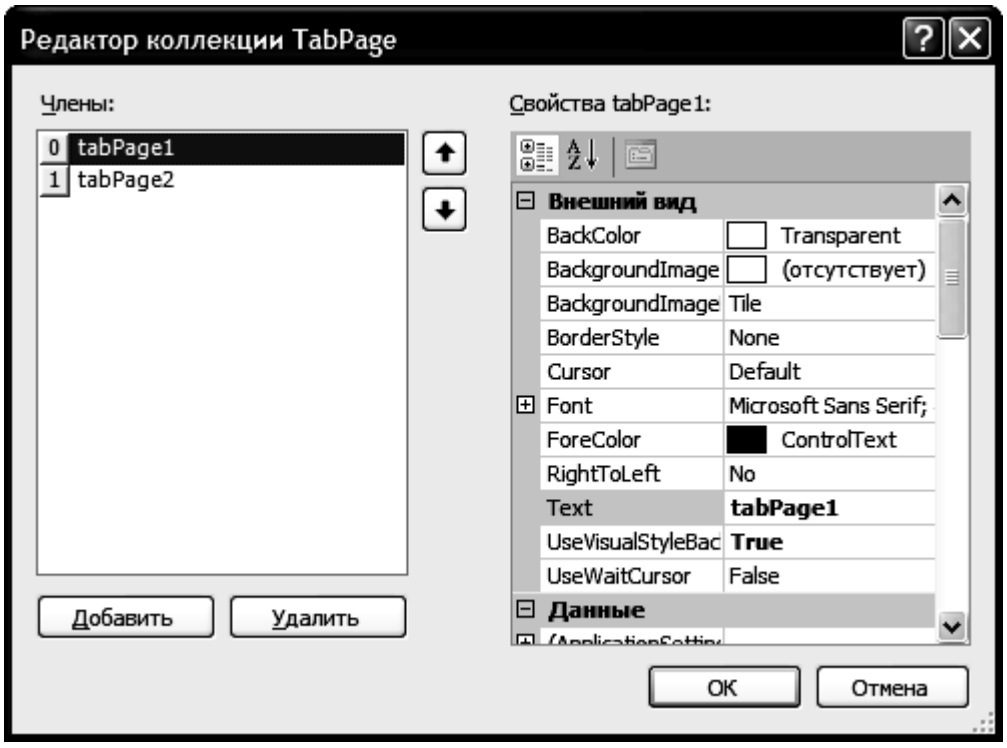


Рисунок 8.8 – Диалоговое окно `Редактор коллекции TabPage`

8.1.4 Создание диалогового окна

Диалоговое окно (`Toolbox`) предлагает несколько стандартных диалогов. Все они предоставляют достаточно большие возможности, но ни одно из них не подходит в настоящем случае. Поскольку данная программа требует выполнения очень специфичных действий, нам придется самостоятельно создать диалоговое окно. Диалоговое окно – это всего лишь форма, значение свойства `FormBorderStyle` которой установлено равным `FixedDialog`, поэтому инструмент *Конструктор форм* значительно облегчит создание диалогового окна. Нужно выбать команду главного меню *Проект→Добавить новый элемент* или нажать комбинацию клавиш <Ctrl+Shift+A>, чтобы открыть диалоговое окно *Добавление нового элемента*, показанное на рис. 8.9.

В списке *Категории*: в левой панели установить категорию `UI` (*Интерфейс пользователя*), а в левой панели *Шаблоны* выберать шаблон *`Форма Windows Forms`* и ввести имя, как показано на рис. 8.9. Щелчок на кнопке *Добавить* приводит к добавлению в проект новой формы и ее отображению в окне *Редактора*.

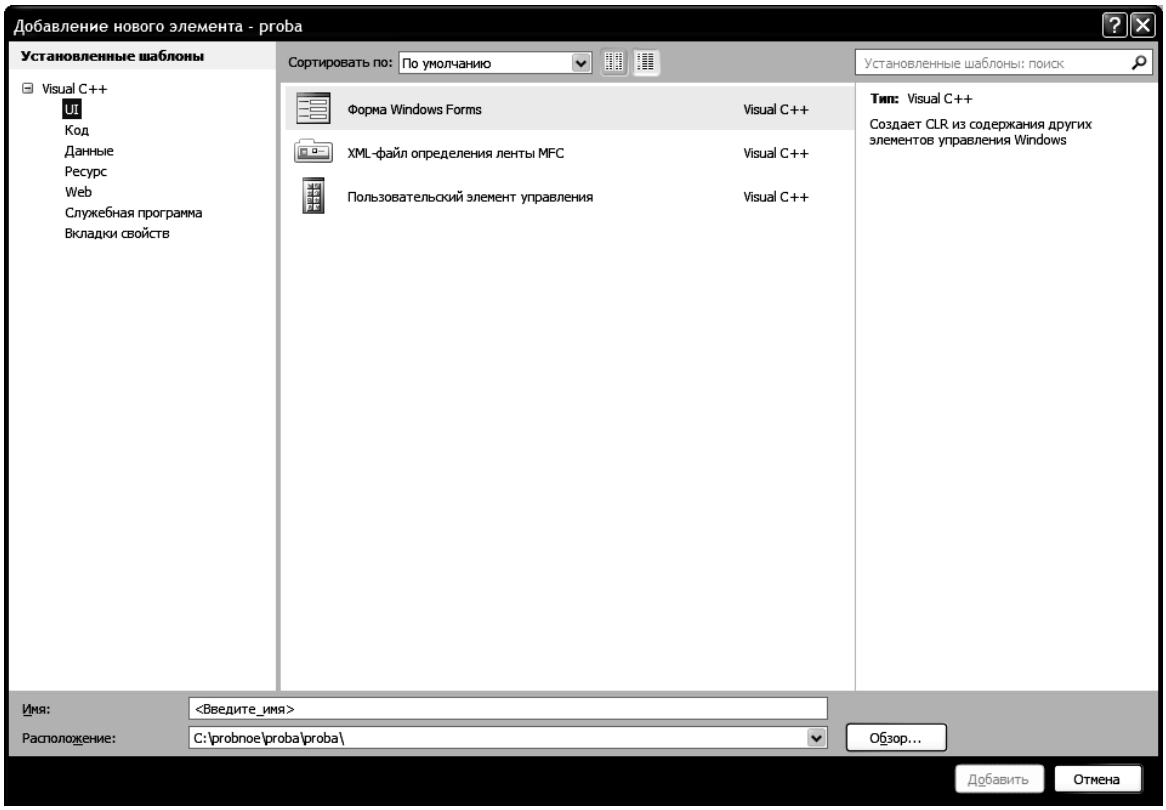


Рисунок 8.9 – Диалоговое окно Добавление нового элемента

Нажать клавишу <F4>, чтобы открыть окно *Свойства* для новой формы. Значение свойства `Text` можно изменить. Этот текст отображается в строке заголовка диалогового окна. Ширину окна можно настроить,

перетаскивая его правый край до тех пор, пока строка заголовка не будет видна полностью. Можно также установить значение свойства Startposition (Начальная позиция) в группе свойств *Макет*, равным Center Parent (По центру родительской формы), чтобы диалоговое окно отображалось в центре отображающей его родительской формы, в данном примере – окна приложения. Поскольку создаваемое окно будет диалоговым окном, а не окном приложения, установить значение свойства FormBorderStyle равным FixedDialog. Во время его отображения диалоговое окно не должно быть доступным для сворачивания или разворачивания на весь экран пользователем, поэтому для удаления из него упомянутых возможностей установите значения свойств MinimizeBox и MaximizeBox в группе WindowStyle, равными False. Диалоговое окно должно закрываться кнопками, которые мы создадим для него, поэтому установить значение свойства ControlBox равным False, чтобы удалить из строки заголовка рамки управляющих и системных элементов.

Следующий шаг – добавление двух кнопок в нижнюю часть формы. Они будут кнопками *OK* и *Cancel* (Отмена) диалогового окна. Для левой кнопки в качестве значения свойства Text установить «*OK»*. Можно также установить равным *OK* значение свойства DialogResult в группе Поведение. Значениями для этих же свойств правой кнопки должны быть, соответственно, «*Cancel*» и *Cancel*. Эффект установки значения свойства DialogResult кнопок заключается в установке значения DialogResult диалогового окна соответствующим этому свойству той кнопки, на которой был выполнен щелчок для закрытия диалогового окна. Это позволяет программно проверять, какая кнопка использовалась для закрытия диалогового окна, и выполнять различный код, в зависимости от того, была ли это кнопка *OK* или *Cancel*.

Теперь, когда мы добавили кнопки в диалоговое окно, можно вернуться к свойствам диалога и установить значения свойств AcceptButton и CancelButton в группе свойств *Разные* равными. В результате при открытом диалоговом окне нажатие клавиши <Enter> клавиатуры будет равносильно щелчку на кнопке *OK*, а нажатие клавиши <Esc> – щелчку на кнопке *Cancel*.

Графический интерфейс пользователя диалогового окна готов, но чтобы он выполнял необходимые действия, придется снова заняться созданием кода.

8.1.5 Создание обработчика событий диалогового окна

Добавить функцию обработчика события Click для кнопки *OK*. Для этого дважды щелкнуть на кнопке *OK*, чтобы добавить каркасный код.

Для кнопки *Cancel* обработчик события Click не нужен. Щелчок на этой кнопке влечет за собой всего лишь закрытие диалогового окна без необходимости выполнения каких-то дальнейших действий.

Значение свойства `DialogResult`, установленное для диалогового окна, определяет, было ли закрыто диалоговое окно. Если в качестве его значения установлено `None`, диалоговое окно не закрыто. Это значение свойства можно устанавливать, если по какой-то причине требуется предотвратить закрытие диалогового окна, например, в случае недопустимости введенного значения.

8.1.6 Помещение на форму диалоговых окон `ColorDialog`, `FontDialog`, `SaveFileDialog`

Диалоговое окно «`ColorDialog`» предназначено для отображения диалогового окна, в котором с помощью мышки можно выбирать цвет для изменения свойств любого из элементов формы (рис. 8.10).

Рисунок 8.10 – Диалоговое окно ColorDialog

Диалоговое окно «`FontDialog`» предназначено для отображения диалогового окна, в котором с помощью мышки можно выбирать параметры шрифта (вид, размер, начертание и т. д.) для изменения свойства Font элементов формы (рис. 8.11).

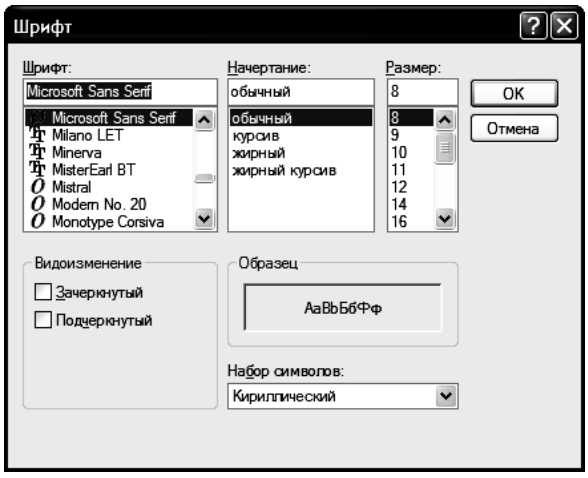


Рисунок 8.11 – Диалоговое окно FontDialog

Диалоговое окно «SaveFileDialog» предназначено для отображения диалогового окна, в котором пользователь может выбирать параметры сохранения (имя файла, тип файла, путь для сохранения) (рис. 8.12).

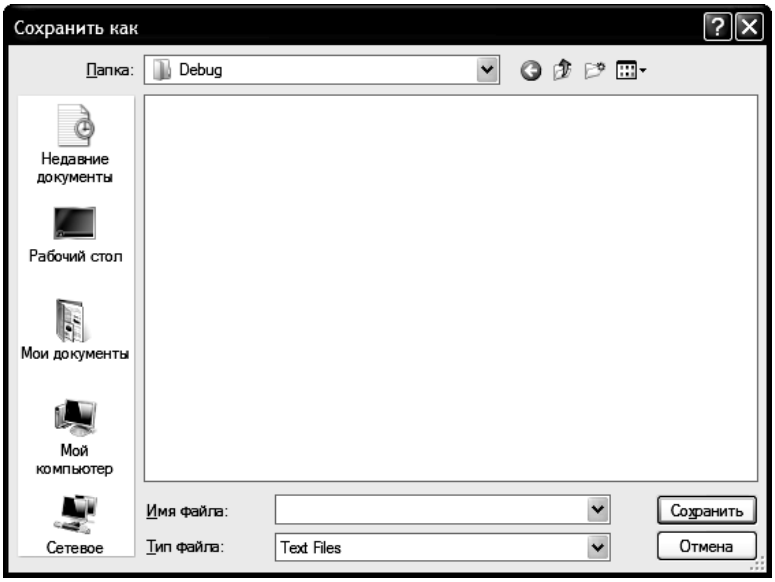


Рисунок 8.12 – Диалоговое окно SaveFileDialog

После размещения в форме диалоговых окон `ColorDialog`, `FontDialog`, `SaveFileDialog` они на форме не появляются, т. е. они являются невизуальными. Они отображаются ниже формы (рис. 8. 13).

Вызов соответствующего окна диалога делается в коде программы.

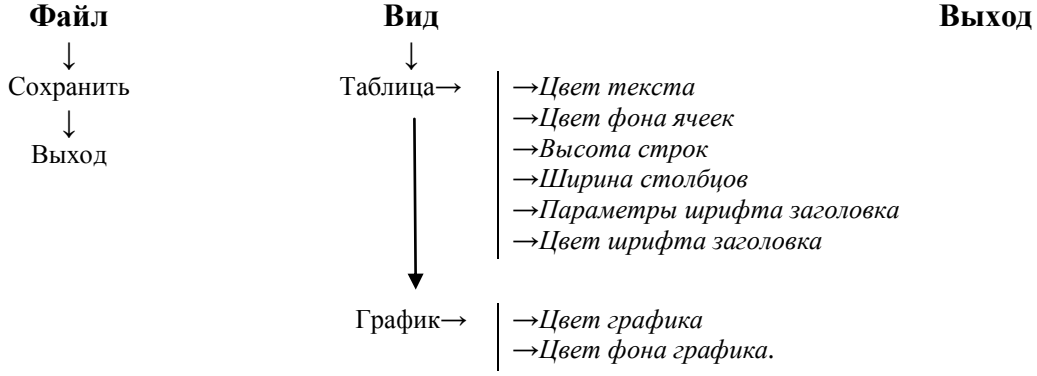
Рисунок 8.13 – Размещение диалоговых окон

Окна *ColorDialog*, *FontDialog*, *SaveFileDialog* применяются только для получения от пользователя желаемых значений настроек. Другими словами, рассматриваемые компоненты – это оболочки, которые представляют пользователю стандартные диалоговые окна, с которыми он может взаимодействовать в хорошо знакомой манере: сгенерированные окна ведут себя так же, как и аналогичные окна в других приложениях ***Windows***. Пользователь управляет содержимым окна путем установки связанных с ним свойств. После того, как окно будет заполнено и закрыто, результат возвращается через одно или несколько свойств в соответствующий компонент. Поэтому после того, как пользователь сделает свой выбор, код приложения должен проверить значения свойств, которые изменились, и каким-то образом отреагировать на действия пользователя: используя значения определенных свойств, выполнить нужные действия. Работа диалогового окна состоит лишь в том, чтобы предоставить пользователю знакомый интерфейс. После закрытия окна необходимо программно проверить

возвращенные им значения и сделать в коде соответствующие действия по сохранению файла. То же самое касается и других стандартных диалоговых окон, поскольку они лишь сообщают приложению о выборе пользователя, но не выполняют никаких самостоятельных действий.

8.2 Пример выполнения работы

Создать Windows-приложение, которое состоит из трех вкладок и операционного меню. На первой вкладке пользователю предлагается ввести четыре параметра (начальное значение интервала, конечное значение интервала, шаг изменения на интервале и параметр **a**), на второй вкладке отображается таблица табулирования функции на интервале и экстремумы этой функции, на третьей вкладке отображается график функции на данном интервале. В верхней части окна есть три пункта операционного меню: Файл, Вид и Выход. Структура меню должна быть следующей:



Ход выполнения

1 Войти в среду ***Visual Studio 2010*.**

2 В окне **Создать Проект** следует развернуть узел Visual C++, обратиться к пункту `CLR` и на центральной панели выбрать *`Приложение Windows Form`*.

3 Затем в поле редактора **Имя** (где по умолчанию имеется <Введите имя>) следует ввести имя проекта, **`Visual_Lab8`**. В поле **Расположение** можно указать путь размещения проекта или выбрать путь размещения проекта с помощью клавиши (кнопки) **Обзор** (например, N:\\CI\2_trim\Lab8).

4 Для формы изменить значение свойства **`Text`**, занеся, например, следующие данные: ***«Выполнил студент группы ЭСА-11-1 Иванов П. А. Лабораторная работа 8».***

5 Свойству **`FormBorderStyle`** формы (стиль рамки окна) присвоить значение ***`FixedToolWindow`***. Это значение определяет окно как диалоговое, его размеры на этапе прогона приложения (в процессе работы приложения) не могут быть изменены.

6 В верхней части формы разместить элемент `MenuStrip` (*операционное меню*) и разместить в нем три пункта: «*Файл*», «*Вид*» и «*Выход*» (*как это сделать, изложено в теоретическом материале в начале лабораторной работы*), а в них подпункты в соответствии с условием задания. Получим следующий вид меню (рис. 8.14–8.16).

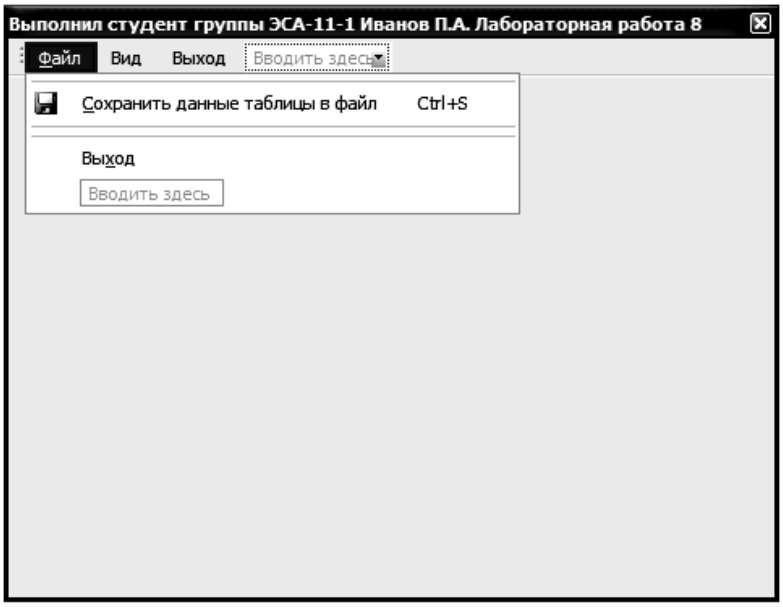


Рисунок 8.14 – Подпункты меню пункта Файл

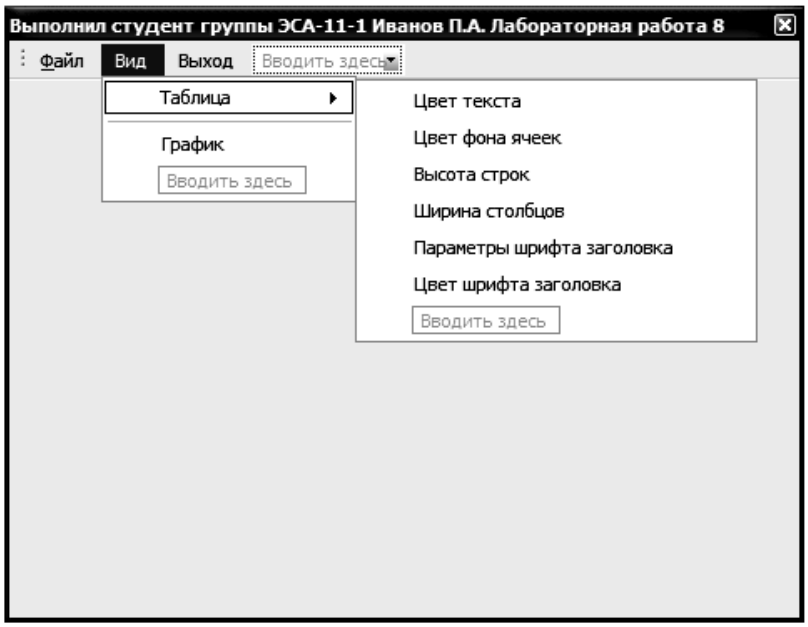


Рисунок 8.15 – Подпункты меню пункта Вид→ Таблица

7 Внизу формы разместить компонент **`Button1`** и в свойство **`Text`** занести текст «***Выход***» (рис. 8.17).

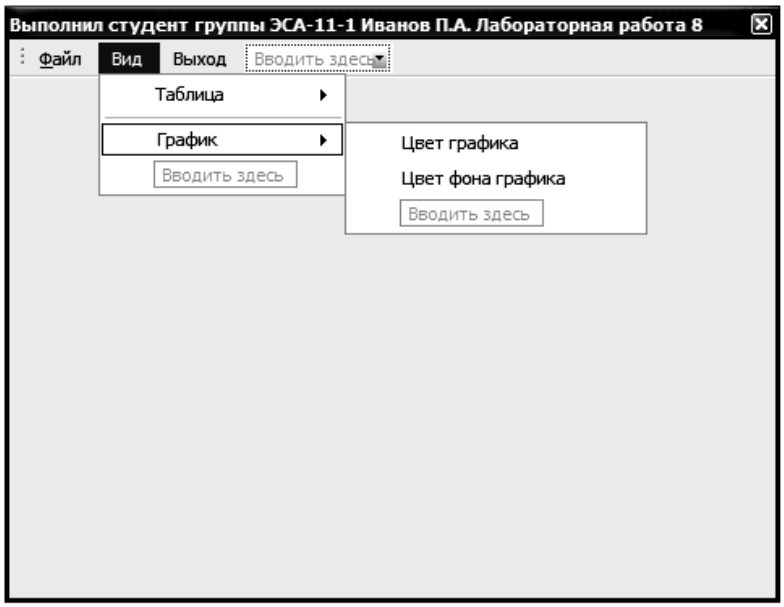


Рисунок 8.16 – Подпункты меню пункта Вид→ График

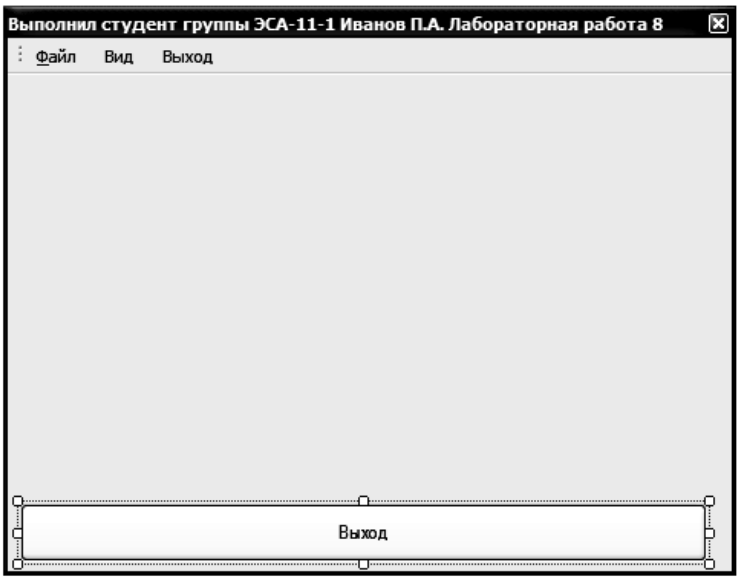


Рисунок 8.17 – Вид формы на этапе проектирования

8 На форму разместить элемент `TabControl` (*теория по этому элементу в начале лабораторной работы*) и получить следующий вид (рис. 8.18).

9 Добавить в компонент `TabControl` еще одну вкладку `TabPage3` и для каждой из вкладок в свойство **`Text`** занести текст на русском языке (рис. 8.19). На первую вкладку `TabPage1` (*Ввод данных*) занести 5 элементов `Label` и 4 элемента `TextBox`. Для компонентов `Label` изменить свойство **`Text`**, в результате проведенных операций должна получиться форма примерно такого вида, как показано на рис. 8.19.

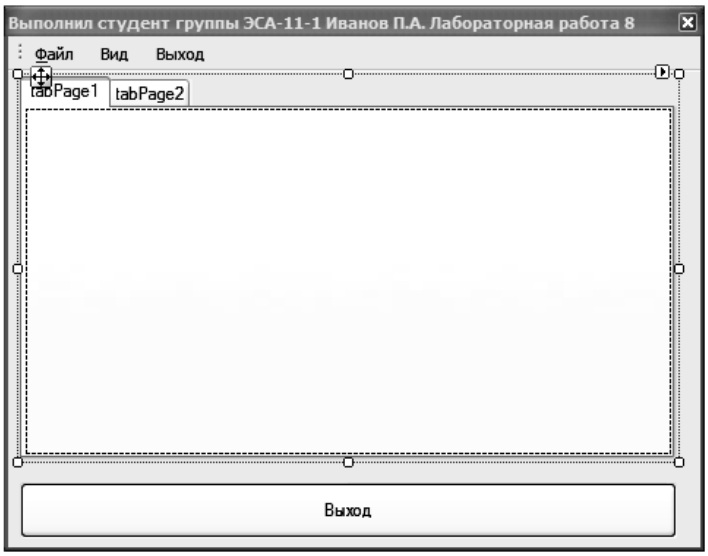


Рисунок 8.18 – Вид формы после занесения на нее элемента `TabControl`

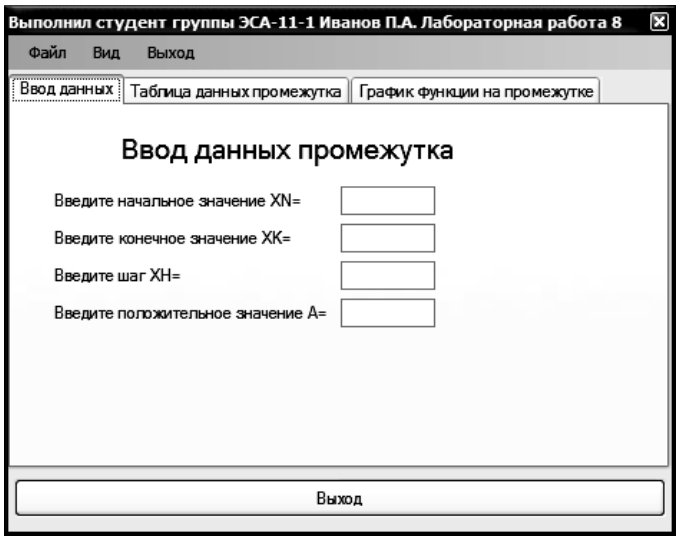


Рисунок 8.19 – Вид вкладки `TabPage1`

10 Перейти на вторую вкладку `TabPage2` (*Таблица данных промежутка*) и на ней разместить 3 элемента `Label`, 2 элемента `TextBox` и один элемент `DataGridView`. В результате проведенных операций должна получиться форма примерно такого вида, как показано на рис. 8.20.

11 Перейти на третью вкладку `TabPage3` (*График функции на промежутке)* и на ней разместить элемент `Chart1` (*настройка этого элемента рассматривалась в предыдущих лабораторных работах*). В результате проведенных операций должна получиться форма примерно такого вида, как показано на рис. 8.21.

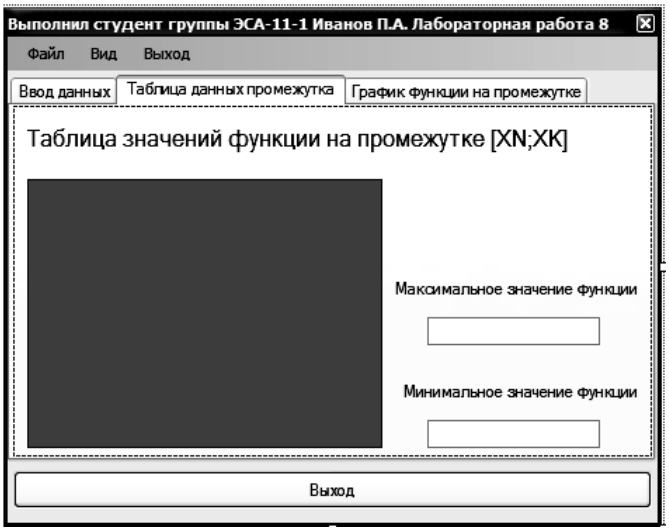


Рисунок 8.20 – Вид вкладки `TabPage2`

Рисунок 8.21 – Вид вкладки `TabPage3`

12 Создать вспомогательную форму (`Form2`) для ввода параметров изменения внешнего вида элемента `DataGridView` главной формы (*процесс создания диалогового окна описан в теоретической части п. 8.1.4*). В результате проведенных операций должна получиться форма примерно такого вида, как показано на рис. 8.22.

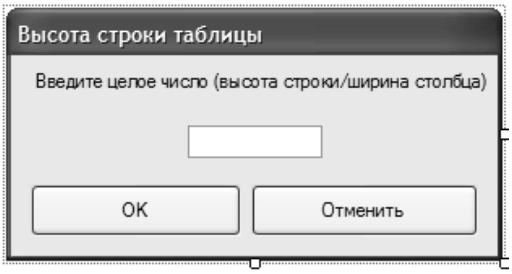


Рисунок 8.22 – Внешний вид `Form2`

13 Перейти к коду программы формы Form1. Сразу после строки:

```
#pragma once
```

внести используемые в программе переменные, подключить библиотеку использования математических функций и подключить вторую форму (Form2), для этого вставитьследующие строки:

```
double xn,xk,xh,x,y,a,ymax,ymin,yt;
                int i,k,j;
                int h;
                int m;
#include <math.h>
#include "Form2.h"
```

14 В разделе ***using*** добавитьбстроку для отображения графика функции:

```
using namespace System::Windows::Forms::DataVisualization::Charting;
```

15 В нашей программе предусмотрено три варианта для выхода из программы: первый – кнопка внизу формы (Form1) с надписью «Выход», второй – в операционном меню пункт «***Выход***», третий – в операционном меню «***Файл***»→ «***Выход***». Для каждого из этих вариантов создадать заготовки подпрограмм, дважды щелкнув левой кнопкой мышки по соответствующему пункту и вставив код:

```
Application::Exit();
```

16 В нашей программе данные вводятся на первой вкладке компонента **TabControl1**, на второй – вывод таблицы значений и экстремумов функции, а на третей – график функции на данном промежутке. И чтобы это происходило при переключении между вкладками, нужно создать событие SelectedIndexChanged. Для этого нужно выделить компонент TabControl1 и перейти в окно «***Свойства***», выбрав вкладку ***«События»***  , и напротив события SelectedIndexChanged сделать двойной щелчок левой кнопкой мыши. В полученную заготовку подпрограммы внести следующий код:

```
double xn,xk,xh,x,y,a,ymax,ymin,yt;
int n,i;
Series^ plot1 = chart1->Series[0];
plot1->Points->Clear();
if ((textBox1->Text!="")&&(textBox2->Text!="")&&(textBox3-
>Text!="")&&(textBox4->Text!="")){
     xn = Convert::ToDouble(textBox1->Text);
     xk = Convert::ToDouble(textBox2->Text);
     xh = Convert::ToDouble(textBox3->Text);
     a = Convert::ToDouble(textBox4->Text);
if ((xn>=xk)||(xh>(xk-xn))) {MessageBox::Show( "Данные запол-
нены неверно", "Ошибка ввода данных", MessageBoxButtons::OK,
MessageBoxIcon::Exclamation );}
else
{x=xn;
dataGridView1->Columns->Clear();
```

```
 //Заполнение DGView столбцами
        dataGridView1->ColumnCount = 2;
 //Заполнение DGView строками
dataGridView1->Rows->Add(ceil((xk-xn)/xh)+1);
dataGridView1->Rows[0]->Cells[0]->Value =Convert::ToString("
x");
dataGridView1->Rows[0]->Cells[1]->Value =Convert::ToString("
Y");
i=1;
x=xn;ymax=-1.8e307;ymin=1.8e307;
while (x<=xk)
          {
     if (x<=0){ y=2*x+2;}
          else
     if (x<=a) {y=sqrt(x+3);}
          else
     if (x>a) {y=pow(cos(x+2),2);}
plot1->Points->AddXY(x, y);
dataGridView1->Rows[i]->Cells[0]->Value=Convert::ToString(x);
//переменной yt присваивает округленное до двух знаков после запятой значение y
yt=ceil(y*100)/100;
//Вывод во втором столбце таблицы значение функции
dataGridView1->Rows[i]->Cells[1]->Value
=Convert::ToString(yt);
//находит максимальное и минимальное значение и округляет до  двух знаков после запятой
     if (y>ymax) ymax=ceil(y*100)/100;
     if (y<ymin) ymin=ceil(y*100)/100;
     x=x+xh;
     i++;}}
//выводит в компоненты textbox максимальное и минимальное значение функции
textBox5->Text = Convert::ToString (ymax);
textBox6->Text = Convert::ToString (ymin);
}
else
{MessageBox::Show( "Заполните, пожалуйста, данные", "Ошибка
ввода данных",MessageBoxButtons::OK,MessageBoxIcon::Exclamation
);}
```

17 Создать событие ***Click*** для пункта операционного меню «***Вид***»→ «***Таблица***»→ «***Цвет текста***» (при нажатии на этот пункт будет появляться диалоговое окно выбора цвета, и после выбора пользователем цвета, цвет текста таблицы будет меняться). Для этого сделать двойной щелчок левой кнопкой мыши по надписи «***Цвет текста***» и в полученную заготовку подпрограммы вставить код:

```
if(colorDialog1->ShowDialog()==System::Windows:
:Forms::DialogResult::OK)
{dataGridView1->DefaultCellStyle->ForeColor =colorDialog1-
>Color;}
```

18 Создать событие ***Click*** для пункта операционного меню «***Вид***»→ «***Таблица***»→ «***Цвет фона ячеек***» (при нажатии на этот пункт будет

появляться диалоговое окно выбора цвета, и после выбора пользователем цвета, цвет фона ячеек таблицы будет меняться). Для этого сделать двойной щелчок левой кнопкой мыши по надписи «***Цвет фона ячеек***» и в полученную заготовку подпрограммы вставить код:

```
if(colorDialog1-
>ShowDialog()==System::Windows::Forms::DialogResult::OK)
      {dataGridView1->DefaultCellStyle-
>BackColor=colorDialog1->Color;}
```

19 Создадть событие ***Click*** для пункта операционного меню «***Вид***»→ «***Таблица***»→ «***Высота строк***» (при нажатии на этот пункт будет появляться диалоговое окно, в которое пользователь вводит целое число, и при нажатии на кнопку «***ОК***» высота строк таблицы меняется). Для этого сделатьдвойной щелчок левой кнопкой мыши по надписи «***Высота строк***» и в полученную заготовку подпрограммы вставить код:

```
Form2^ f = gcnew Form2();
this->Hide();
f->ShowDialog();
this->Show();
for (int i = 0; i < dataGridView1->RowCount; i++)
{DataGridViewRow^ row = dataGridView1->Rows[ i ];
row->Height = h;}
```

соответственно для этого пункта меню будет задействовано окно формы (`Form2`), поэтому нужно перейти в код `Form2.h` и вначале прописать использованные переменные в этой форме:

```
extern int h;
extern int m;
```

сделать событие ***Click*** для кнопки с надписью «***ОК***» формы «`Form2`» и в полученную заготовку вставить следующий код:

```
h=Convert::ToInt32(textBox1->Text);
m=Convert::ToInt32(textBox1->Text);
```

сделать событие ***Click*** для кнопки с надписью «***Отменить***» формы «`Form2`» и в полученную заготовку вставить следующий код:

```
this->Hide();
```

20 Создать событие ***Click*** для пункта операционного меню «***Вид***»→ «***Таблица***»→ «***Ширина столбцов***» (при нажатии на этот пункт будет появляться диалоговое окно, в которое пользователь вводит целое число, и при нажатии на кнопку «***ОК***» ширина столбцов таблицы меняется). Для этого сделать двойной щелчок левой кнопкой мыши по надписи «***Ширина столбцов***» и в полученную заготовку подпрограммы вставить код:

```
Form2^ f = gcnew Form2();
     this->Hide();
     f->ShowDialog();
     this->Show();
for (int i = 0; i < dataGridView1->ColumnCount; i++)
     {DataGridViewColumn^ column = dataGridView1->Columns[ i ];
      column->Width = m;}
```

21 Создать событие ***Click*** для пункта операционного меню «***Вид***»→ «***Таблица***»→ «***Параметры шрифта заголовка***» (при нажатии на этот пункт будет появляться диалоговое окно «***Шрифт***», в котором пользователь выбирает параметры шрифта (вид шрифта, размер, начертание и т. д.), и при нажатии на кнопку «***ОК***» внешний вид заголовка таблицы меняется). Для этого сделать двойной щелчок левой кнопкой мыши по надписи «***Параметры шрифта заголовка***» и в полученную заготовку подпрограммы вставим код:

```
if(fontDialog1-
>ShowDialog()==System::Windows::Forms::DialogResult::OK)
{label6->Font = fontDialog1->Font;}
```

22 Создадать событие ***Click*** для пункта операционного меню «***Вид***»→ «***Таблица***»→ «***Цвет шрифта заголовка***» (при нажатии на этот пункт будет появляться диалоговое окно «***ColorDialog***», в котором пользователь выбирает цвет шрифта, и при нажатии на кнопку «***ОК***» внешний вид заголовка таблицы меняется). Для этого сделать двойной щелчок левой кнопкой мыши по надписи «***Цвет шрифта заголовка***» и в полученную заготовку подпрограммы вставить код:

```
if(colorDialog1->ShowDialog()==System:
:Windows::Forms::DialogResult::OK)
{label6->ForeColor = colorDialog1->Color;}
```

23 Создать событие ***Click*** для пункта операционного меню «***Вид***»→ «***График***»→ «***Цвет графика***» (при нажатии на этот пункт будет появляться диалоговое окно «***ColorDialog***», в котором пользователь выбирает цвет, и при нажатии на кнопку «***ОК***» внешний вид графика меняется). Для этого сделать двойной щелчок левой кнопкой мыши по надписи «***Цвет графика***» и в полученную заготовку подпрограммы вставить код:

```
Series^ plot1 = chart1->Series[0];
if(colorDialog1->ShowDialog()==System:
:Windows::Forms::DialogResult::OK)
plot1->Color = colorDialog1->Color;
```

24 Создать событие ***Click*** для пункта операционного меню «***Вид***»→ «***График***»→ «***Цвет фона графика***» (при нажатии на этот пункт будет появляться диалоговое окно «***ColorDialog***», в котором пользователь выбирает цвет, и при нажатии на кнопку «***ОК***» внешний вид графика меняется). Для этого сделать двойной щелчок левой кнопкой мыши по надписи «***Цвет фона графика***» и в полученную заготовку подпрограммы вставить код:

```
Series^ plot1 = chart1->Series[0];
if(colorDialog1->ShowDialog()==System:
:Windows::Forms::DialogResult::OK)
{chart1->BackColor = colorDialog1->Color;}
```

25 Создать событие ***Click*** для пункта операционного меню «***Файл***»→ «***Сохранить данные таблицы в файл***» (при нажатии на этот пункт будет появляться диалоговое окно «***SaveFileDialog***», в котором пользователь указывает имя и выбирает путь для сохранения файла

с данными таблицы, и при нажатии на кнопку «***ОК***» данные таблицы автоматически заносятся в новый созданный текстовый файл). Для этого сделать двойной щелчок левой кнопкой мыши по надписи «***Сохранить данные таблицы в файл***» и в полученную заготовку подпрограммы вставить код:

```
SaveFileDialog ^saveFileDialog1 = gcnew SaveFileDialog();
saveFileDialog1->Filter = "Text Files|*.txt"  ;
saveFileDialog1->FilterIndex = 2 ;
saveFileDialog1->RestoreDirectory = true ;
if(saveFileDialog1->ShowDialog() == System::Windows::Forms:
:DialogResult::OK)
 {
StreamWriter^ pwriter = gcnew StreamWriter(saveFileDialog1-
>FileName);
for (int i = 0; i < dataGridView1->RowCount; i++)
pwriter->WriteLine(dataGridView1->Rows[i]->Cells[0]->Value-
>ToString()+"   |   "+dataGridView1->Rows[i]->Cells[1]->Value-
>ToString());
pwriter->Close();}
```

26 На примере лабораторной работы 2 сделать проверку на корректность ввода данных (XN, XK, XH и A), используя событие **`Leave`** в окне свойств (в код программы добавить 4 процедуры, проверяющие правильность ввода).

27 Запустить программу на выполнение, нажав на функциональную кнопку **F5**. Получим следующий вид окна рис. 8.23–8.25.

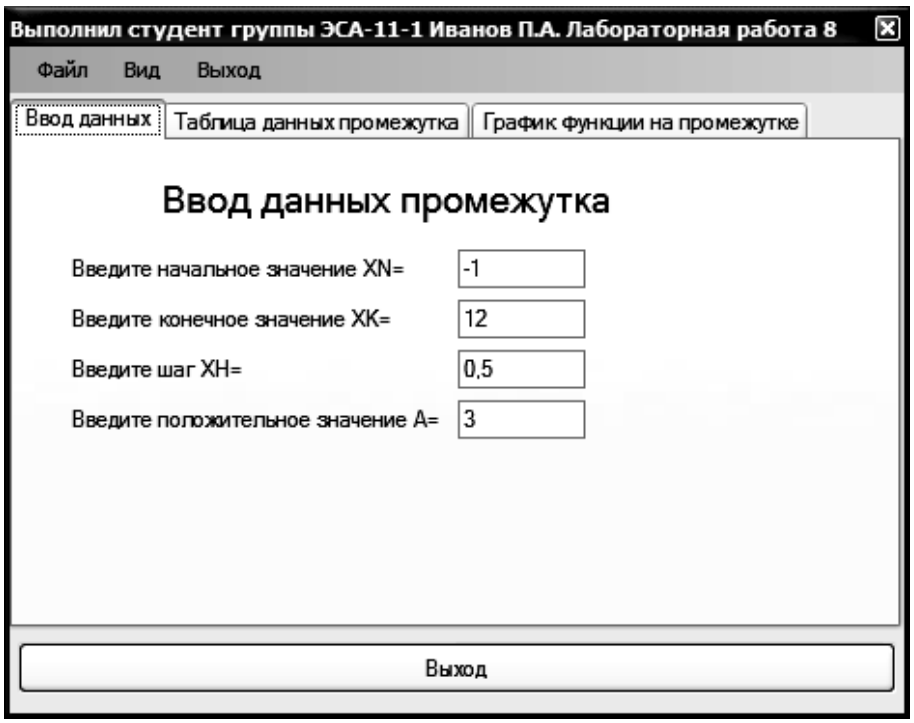


Рисунок 8.23 – Рабочий вид формы приложения вкладки «Ввод данных»

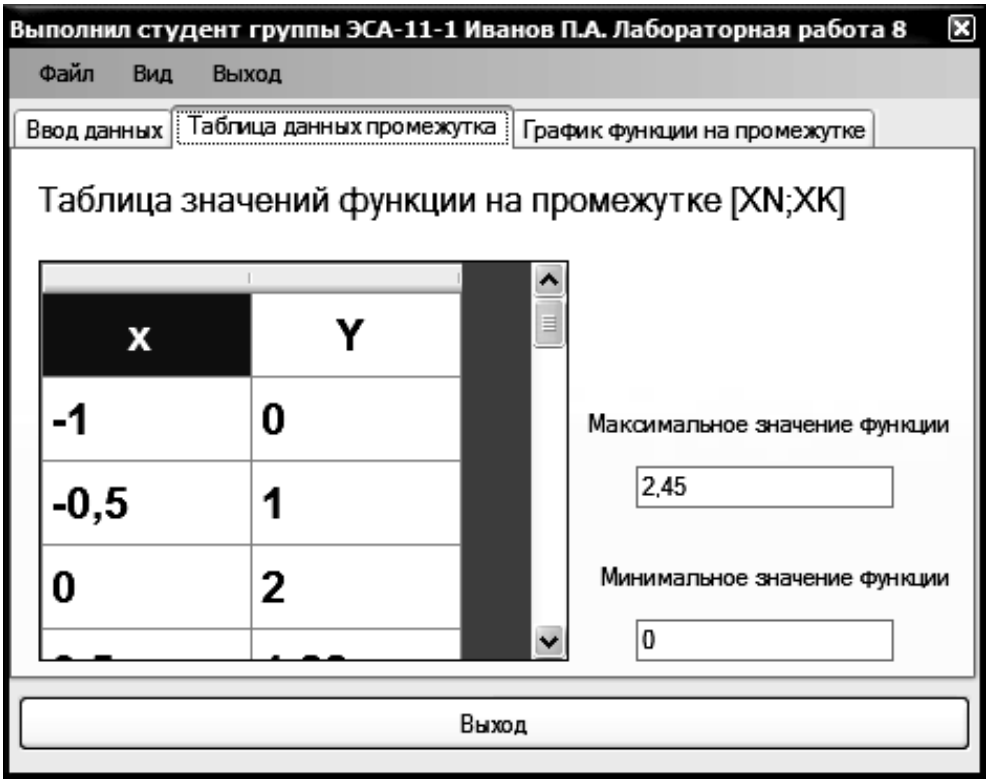
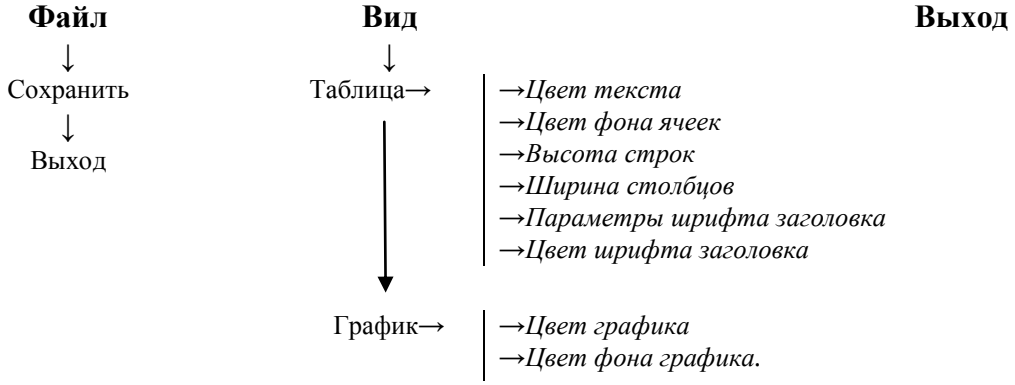


Рисунок 8.24 – Рабочий вид формы приложения вкладки «Таблица данных промежутка»

Рисунок 8.25 – Рабочий вид формы приложения вкладки «График функции на промежутке»

8.3 Рабочее задание

Создать Windows-приложение, которое состоит из трех вкладок и операционного меню. На первой вкладке пользователю предлагается ввести четыре параметра (начальное значение интервала, конечное значение интервала, шаг изменения на интервале и параметр **a**), на второй вкладке отображается таблица табулирования функции на интервале и экстремумы этой функции, на третьей вкладке отображается график функции на данном интервале. В верхней части окна есть три пункта операционного меню: Файл, Вид и Выход. Все пункты операционного меню должны быть рабочими. Структура меню должна быть следующей.



Данные функции, промежутка и значения **a** взять из таблицы 8.1.

Таблица 8.1 – Индивидуальные задания

Вариант	Функции			Значение параметра **a**	Границы отрезка	Шаг табулирования
	$f1(x)$	$f2(x)$	$f3(x)$			
1	*2*	*3*	*4*	*5*	*6*	*7*
1	$x^5 \operatorname{ctg}(2x^3)$	$\frac{5}{\operatorname{tg}(2x+3)+1}$	$x^2 e^{-x}$	2	[–1,6;3,7]	0,3
2	$x^x \operatorname{tg}(x+5)$	$x^3 \cos x$	$\sin x^2 + x^{0.25}$	5	[–2,8; 8,2]	0,4
3	$\ln\lvert x+1\rvert + \sqrt{3\lvert x\rvert}$	$\sin(x^2+3x)$	$\ln^2\lvert x\rvert + \sqrt{x+3}$	1	[–1,7; 2,6]	0,25
4	$\sin^2 x^3$	$\sqrt[5]{6x-x^2+1}$	$\sin(x-e^{-x})$	3	[–2,2; 7,4]	0,23

Продолжение талицы 8.1

1	*2*	***3***	*4*	*5*	*6*	*7*
5	$\cos\left(x^3+1\right)e^{-x}$	$\left\|\sqrt[3]{\dfrac{2x+5}{\left(x^3+2\right)}}\right\|$	$2\sqrt{x^2+7\sin\left(x^3\right)}$	4	[–1,1; 7,9]	0,8
6	$2\sqrt{\left\|x^3\right\|}\sin\left(x^3\right)$	$(x+1)^2\cos x^3$	$\sqrt{x^4+2}+\sin x^2$	1	[–1,2; 2,6]	0,1
7	$\sin\left(x^5+3\right)$	$\sqrt{x^3}\sin x$	$x^4-\sin\left(x+1\right)$	1	[–1,7; 2,4]	0,3
8	$x^4\mathrm{tg}\left(x+2\right)$	$\ln\left(4x^2+1\right)$	$\ln\sqrt[5]{5+x^2}$	5	[–4,3; 8,0]	0,5
9	$x^5+\sqrt[3]{x+10}$	$1{,}3\sqrt{4+x^2}$	$\left\|x+1\right\|^x$	3	[–9,1; 5,8]	0,14
10	$\left\|x\right\|^5\mathrm{ctg}\left\|2x\right\|$	$\ln\left(x^2+1\right)$	$e^{-2x}-\sqrt[3]{\left\|x+1\right\|}$	1	[–3,4; 2,5]	0,23
11	$x^5\mathrm{ctg}\left(2x^3\right)$	$\sqrt[5]{x^4+3}$	$\left\|\sin^2 x+1\right\|^{2x}$	3	[–2,2; 8,1]	0,15
12	$\mathrm{ctg}\left(3x-1\right)^2$	$2+xe^{-x}$	$\sin\left(x^3+1\right)$	2	[–2,8; 5,2]	0,5
13	$x^3+4x^2\sqrt{\left\|x\right\|}$	$(x-1)^3+\cos x^3$	$\sqrt{\left\|x\right\|^3}\sin x^3$	4	[–3,2; 7,8]	0,36
14	$(2x+1)(\left\|x\right\|+2)^3$	$e^x+\sin\left(x+2\right)$	$3\ln\sqrt[5]{\sin^2 x+2}$	1	[–6,1; 1,3]	0,15
15	$\mathrm{ctg}\left(x^3+1\right)$	$\ln\left(\sin x+1\right)^2$	$\sqrt[3]{2x^2+x^4+1}$	2	[–7,4; 3,6]	0,6
16	$1{,}3\sqrt{4+x^2}$	3^{x+3}	$5^{x+1}+\mathrm{tg}\left(x+1\right)$	4	[–1,2; 7,1]	0,45
17	$e^{2x}+\sin\left(2x^3\right)$	$\sin^3 x^4$	$e^{-x}+\sqrt[3]{3x^2+1}$	2	[–2,2; 3,9]	0,55
18	$x^3+(\left\|x\right\|+1)^{0,1x}$	$\left(x-1\right)^3+\cos\left(x^3\right)$	$2x+\mathrm{tg}\left(x^2+2\right)$	2	[–0,3; 4,5]	0,62
19	$\left\|\sqrt[3]{\dfrac{2x+5}{\left(x^3+2\right)}}\right\|$	$\dfrac{5x+x^2}{\left(x^2+3\right)^3}$	$\cos^2\left(x^3+\sqrt{x}\right)$	2	[–2,4; 4,4]	0,4
20	$\ln\left(x^2+5\right)$	$\sin\left(e^x+2\right)$	$\dfrac{\sin\left(x+3\right)}{e^{2x}+\cos\left(x+1\right)}$	5	[–2,9; 8,2]	0,2
21	$\sqrt[5]{x^2+x+1}$	$\ln^2\left(\left\|\sqrt{x+5}\right\|\right)$	$\sin\left(x^2\right)+x^{0,25}$	2	[–3,9; 3,8]	0,15
22	$3x^5+\mathrm{ctg}\left(x^3+1\right)$	$e^{x+1}-\sin\left(\pi x\right)$	$\sqrt[5]{\sin^2 x+2}$	3	[–1,3; 7,1]	0,6
23	$x^5-\mathrm{ctg}\left(\pi x^3\right)$	$(\left\|7x\right\|+1)^{0,3}+\sin x$	$5x-x^2$	2	[–2,9; 6,2]	0,8

Продолжение талицы 8.1

1	*2*	*3*	*4*	*5*	*6*	*7*
24	$\lvert x\rvert^{x+2}+\sin(x)$	$3^{x+3}+2x$	$\sqrt[5]{x^2+x+1}$	4	[–3,7; 8,5]	0,11
25	$x^2+\sin(7x)-1$	$\lvert x^3+10^x\rvert$	$\sqrt[7]{2x^4+x^2+1}$	1	[–3,9; 1,2]	0,25
26	$\sqrt[3]{\lvert x\rvert+2}-1$	$\sin(x^2)+x^{0,25}$	$\ln^2(x)+\sqrt{x}$	4	[–4,5; 6,1]	0,3
27	$\sqrt{\lvert\sin^2 x+\cos^4 x\rvert}$	$\ln(x+1)+\sqrt{3x}$	$5^{x+1}+\operatorname{tg}(x+3)^2$	2	[–3,4; 3,4]	0,33
28	$x^3-3x^2\sqrt{\lvert x\rvert+6}$	$\dfrac{2x+2}{\operatorname{tg}(2x-1)+1}$	x^4-x^x	3	[–4,1; 5,0]	0,45
29	x^4+2x^3-x	$e^{-x}+\sqrt[3]{x}$	$\ln(\lvert x^3+x^2\rvert)$	1	[–1,7; 2,9]	0,75
30	$\dfrac{(3x-1)^2}{x^5+2x+1}$	$\ln^2\lvert\sqrt{x+5}\rvert$	$\sqrt[5]{1+x^2}$	2	[–1,6; 4,7]	0,65

8.4 Контрольные вопросы

1 Компонент `MenuStrip`, его свойства.

2 Диалоговое окно `ColorDialog`, его вызов и назначение.

3 Диалоговое окно `FontDialog`, его вызов и назначение.

4 Диалоговое окно `SaveDialog`, его вызов и назначение.

5 Создание модального диалогового окна, его настройка.

6 Настройка компонента `TabControl`, основное событие для этого элемента.

9 САМОСТОЯТЕЛЬНАЯ РАБОТА. ОБРАБОТКА МАТРИЦЫ. ФОРМИРОВАНИЕ ОДНОМЕРНЫХ МАССИВОВ ИЗ ДВУМЕРНЫХ. СОРТИРОВКА ОДНОМЕРНЫХ МАССИВОВ

Цель: изучение использования вложенных циклических операторов для ввода, вывода и обработки элементов матрицы; обработка элементов столбцов, строк матрицы. Сортировка одномерных массивов.

9.1 Теоретические сведения

9.1.1 Особенности работы с матрицами

В реальной жизни структуры, соответствующие двумерным массивам в программировании, встречаются достаточно часто. Например, оценки студентов группы по разным контрольным точкам по определенной дисциплине, таблица заработанных футбольной командой очков в чемпионате страны и т. д. При этом появляется необходимость для «жизненного» двумерного массива осуществлять подсчет сумм, произведений, количеств определенных (или всех) элементов по строкам (или по столбцам) исходной матрицы. Например, итоговая оценка студента по дисциплине получается путем суммирования оценок по всем контрольным точкам, т. е. необходимо найти суммы элементов строк, чтобы определить оценки студентов всей группы.

Рассмотрим решение задачи: найти сумму четных элементов каждой строки матрицы A[n][m], где n, m задаются пользователем (количество строк и столбцов матриц), результат записать в массив B.

Нас будет интересовать только сам процесс нахождения сумм четных элементов. Пример фрагмента программы, решающий данную задачу:

```
for (i=0; i<n; i++) //перебираем все строки массива
{ s=0;    //для каждой строки заново начинаем считать сумму
for (j=0; j<m; j++)  //перебираем все элементы данной строки
//находим сумму интересующих нас элементов
if (A[i][j]%2==0) s=s+A[i][j];
B[i]=s;   //записываем полученную сумму в одномерный массив
}
```

Аналогичным образом можно подсчитать и сумму четных элементов столбцов. Для этого необходимо в предыдущем фрагменте программы поменять местами строки, организовывающие работу циклов (`for …`), и при записи полученных сумм в одномерный массив в качестве индекса использовать переменную j (B[j]=s).

9.1.2 Сортировка одномерных массивов

Сортировка представляет собой процесс упорядочения элементов в массиве в порядке возрастания или убывания их значений. Например, массив B из n элементов будет отсортирован в порядке возрастания значений его элементов, если

$$B[1] \le B[2] \le \ldots \le B[n],$$

и в порядке убывания, если

$$B[1] \ge B[2] \ge \ldots \ge B[n].$$

В повседневной жизни процесс сортировки «массивов» используется очень часто. Например, список в журнале студенческой группы (сортировка по алфавиту фамилий, а при необходимости и имен, и отчеств) или итоговая таблица чемпионата страны по футболу (сортировка по набранным командой очкам).

Рассмотрим некоторые из алгоритмов сортировки. При этом будем рассматривать сортировку по возрастанию. Ключевым отличием сортировки по убыванию будет то, что в проверке знак «больше» необходимо будет заменить на знак «меньше».

9.1.2.1 Сортировка методом «пузырька». Сортировка методом «пузырька» основана на выполнении в цикле операций сравнения и при необходимости обмена соседних элементов. Рассмотрим этот метод более подробно.

Сравним первый элемент массива со вторым, если первый окажется больше второго, то поменяем их местами. Затем сравним второй с третьим, если второй окажется больше третьего, то также поменяем их местами. Продолжаем сравнивать попарно соседние элементы до сравнения (n–1)-го и n-го элементов. В результате этих действий самый большой элемент станет на последнее (n-е) место. Теперь повторим данный алгоритм сначала, с 1-го до (n–1)-го элемента. Последний n-й элемент не рассматривается, т. к. он уже занял свое место. После проведения данной операции самый большой элемент оставшейся части массива станет на свое (n–1)-е место. Так, уменьшая количество проверяемых элементов, повторяем до тех пор, пока не проверим последние (точнее первые) два элемента.

Пошаговый пример выполнения данного алгоритма сортировки представлен в таблице 9.1.

Таблица 9.1 – Сортировка методом «пузырька»

Номер элемента	**1**	**2**	**3**	**4**	**5**
Исходный массив	7	3	5	4	2
Первый просмотр	3	5	4	2	7
Второй просмотр	3	4	2	5	7
Третий просмотр	3	2	4	5	7
Четвертый просмотр	2	3	4	5	7

Ниже приведен текст фрагмента программы, предназначенный для сортировки одномерного массива B[n] методом «пузырька».

```
for (j= n-1; j>0; j--)
    for (i= 0; i<j; i++)
        if (B[i]>B[i+1])  {t=B[i]; B[i]=B[i+1]; B[i+1]=t;}
```

9.1.2.2 Сортировка методом перестановки. Сортировка методом перестановки основана на поиске наибольшего элемента среди нескольких и последующего его перемещения на последнюю позицию. Рассмотрим этот метод более подробно.

Найдем в массиве самый большой элемент (при этом запоминаем и номер позиции, где стоит этот элемент) и поменяем его с последним n-м. Самый большой элемент массива станет на свое место. Найдем среди (n–1) первых элементов наибольший и поменяем его местами с (n–1)-м. Повторяем эти действия, уменьшая количество проверяемых элементов на единицу до тех пор, пока количество рассматриваемых элементов не станет равным одному.

Пошаговый пример выполнения данного алгоритма сортировки представлен в таблице 9.2.

Таблица 9.2 – Сортировка методом перестановки

Номер элемента	**1**	**2**	**3**	**4**	**5**
Исходный массив	7	2	5	4	3
Первый просмотр	3	2	5	4	7
Второй просмотр	3	2	4	5	7
Третий просмотр	3	2	4	5	7
Четвертый просмотр	2	3	4	5	7

Ниже приведен текст фрагмента программы, предназначенный для сортировки одномерного массива B[n] методом перестановки.

```
for (j= n-1; j>0; j--)
     { max= B[0]; imax= 0;
       for (i= 1; i<j+1; i++)
            if (B[i]>max)  {max= B[i]; imax= i;}
       t= B[imax]; B[imax]= B[j]; B[j]= t;
      }
```

9.1.2.3 Сортировка методом вставки. Сортировка методом вставки основана на поочередном переносе элементов исходного массива в другой, при этом в новый массив элементы добавляются уже в упорядоченном виде. Рассмотрим этот метод более подробно.

Данный метод удобнее рассмотреть на наглядном примере. Предположим, что вы перевозите многотомное собрание сочинений А. Дюма своей библиотеки на новую квартиру. Необходимо расставить хаотично упакованные в коробку (массив B) книги на книжную полку (массив C)

по порядку. Первую взятую книгу из коробки (первый элемент массива B) помещаем на полку в крайнюю левую позицию (на первую позицию в массив C). Для второй, взятой из коробки, книги делаем проверку, учитывая номер тома (значение элемента массива). Если номер тома взятой книги больше, чем номер тома выставленной на полку книги, то вставляем новую книгу справа от уже стоящей, иначе – слева. Повторяем подобные операции несколько раз. Взяв из коробки k-ю книгу, сравниваем ее (по номеру тома) с уже стоящими на полке справа налево. Если стоящая на полке крайняя правая книга больше по номеру тома, чем взятая из коробки, то сдвигаем книгу на полке правее и проверяем следующую книгу. Найдя позицию расположения взятой из коробки книги среди уже стоящих, вставляем ее на книжную полку. Повторяем данные операции, пока все книги не займут свои позиции на книжной полке.

Пошаговый пример выполнения данного алгоритма сортировки представлен в таблице 9.3.

Таблица 9.3 – Сортировка методом вставки

Первый шаг										
B	8	6	1	3	4	5	9	10	7	2
C	8									
Второй шаг										
B	8	6	1	3	4	5	9	10	7	2
C	6	8								
Третий шаг										
B	8	6	1	3	4	5	9	10	7	2
C	1	6	8							
Четвертый шаг										
B	8	6	1	3	4	5	9	10	7	2
C	1	3	6	8						
Пятый шаг										
B	8	6	1	3	4	5	9	10	7	2
C	1	3	4	6	8					
Шестой шаг										
B	8	6	1	3	4	5	9	10	7	2
C	1	3	4	5	6	8				
и т.д.										

Ниже приведен текст фрагмента программы, предназначенный для сортировки одномерного массива B[n] методом вставки.

```
C[0]= B[0];
for (i= 1; i< n; i++)
     { j= i;
       while (j>=0 && B[i]<C[j-1])
             { C[j]= C[j-1]; j--;}
       C[j]= B[i];}
```

9.1.2.4 Сортировка методом Шелла. Сортировка Шелла была названа в честь её изобретателя – Дональда Шелла, который опубликовал этот алгоритм в 1959 году.

Сортировка Шелла (англ. `Shell sort`) – алгоритм сортировки, являющийся усовершенствованным вариантом сортировки методом «пузырька». Идея метода Шелла состоит в сравнении элементов, стоящих не только рядом, но и на определённом расстоянии друг от друга. Иными словами – это сортировка методом «пузырька» с предварительными «грубыми» проходами.

При сортировке Шелла сначала сравниваются и сортируются между собой значения, отстоящие один от другого на некотором расстоянии d. После выполнения интервал уменьшается по формуле d(k) = [d(k + 1) – 1] / 2. После этого процедура повторяется для уменьшенных значений d, а завершается сортировка Шелла упорядочиванием элементов при d = 1 (то есть обычной сортировкой методом «пузырька»).

Эффективность сортировки Шелла в определённых случаях обеспечивается тем, что элементы «быстрее» встают на свои места.

Ниже приведен текст фрагмента программы, предназначенный для сортировки одномерного массива B[n] методом Шелла.

```
d=n-1;
while (d>0)
     {
       for (j=1; j<n; j+=d)
           { for (i=0; i<n-j; i++)
                { if (B[i]>B[i+1])
                     { t=B[i]; B[i]=B[i+1]; B[i+1]=t; }
                }
           }
       d=(d-1)*0.5;
     }
```

9.2 Рабочее задание

Создать Windows-приложение (табл. 9.4), которое по запрашиваемому количеству строк и столбцов формирует случайным образом двумерный массив A, подсчитывает <условие своего варианта> по строкам (для нечетных вариантов) или по столбцам (для четных вариантов) получившейся матрицы, полученные результаты заносит в одномерный массив B и сортирует массив B методом <условие своего варианта>.

Приблизительный внешний вид создаваемого Windows-приложения изображен на рисунке 9.1.

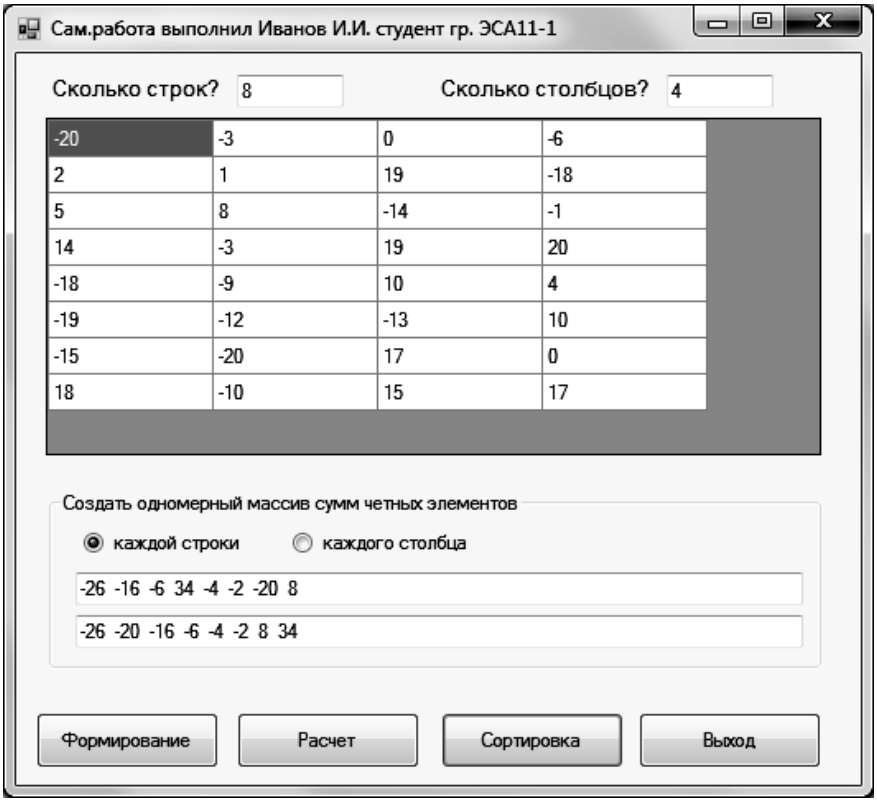


Рисунок 9.1 – Окно формы программы

Таблица 9.4 – Индивидуальные задания

Номер варианта	Условие селекции в двумерном массиве	Метод сортировки
1	*2*	*3*
1	Сумму положительных элементов	«пузырька»
2	Количество четных элементов	перестановки
3	Произведение кратных 3 элементов	вставки
4	Среднее арифметическое отрицательных элементов	Шелла
5	Максимальный элемент	«пузырька»
6	Среднее геометрическое нечетных элементов	перестановки
7	Количество неотрицательных элементов	вставки
8	Произведение четных элементов	Шелла
9	Сумму кратных 5 элементов	«пузырька»
10	Наибольший по модулю элемент	перестановки
11	Среднее арифметическое четных элементов	вставки
12	Среднее геометрическое кратных 4 элементов	Шелла
13	Сумму отрицательных элементов	«пузырька»
14	Количество нечетных элементов	перестановки

Продолжение таблицы 9.4

1	*2*	*3*
15	Произведение некратных 4 элементов	вставки
16	Среднее арифметическое положительных элементов	Шелла
17	Минимальный элемент	«пузырька»
18	Среднее геометрическое четных элементов	перестановки
19	Количество неположительных элементов	вставки
20	Произведение нечетных элементов	Шелла
21	Сумму некратных 3 элементов	«пузырька»
22	Наименьший по модулю элемент	перестановки
23	Среднее арифметическое нечетных элементов	вставки
24	Среднее геометрическое некратных 3 элементов	Шелла
25	Сумму наибольшего и наименьшего элементов	«пузырька»

Контрольные вопросы

1 В чем заключается сортировка элементов одномерных массивов?
2 Сортировка методом «*пузырька*».
3 Сортировка методом *перестановки*.
4 Сортировка методом *вставки*.
5 Сортировка методом *Шелла*.

СПИСОК ЛИТЕРАТУРЫ

1 **Архангельский, А. Я.** Программирование в C++ Builder 6 / А. Я. Архангельский. – М. : Бином, 2003. – с. 368.

2 **Водовозов, В. М.** Конструирование приложений для Windows : учебное пособие / В. М. Водовозов, А. К. Пожидаев. – СПб. : Изд-во СПбГЭТУ «ЛЭТИ», 2004. – с. 412.

3 **Водовозов, В. М.** Объектно-ориентированное программирование на C++ : учебное пособие. / В. М. Водовозов, Ф. В. Чмиленко. – СПб. : Изд-во СПбГЭТУ «ЛЭТИ», 2007. –с. 387.

4 **Коплиен, Дж.** Программирование на C++ / Дж. Коплиен. – СПб. : ПИТЕР, 2005. – 624 с.

5 **Лаптев, В. В.** C++. Объектно-ориентированное программирование / В. В. Лаптев. – СПб. : Питер, 2008. – 389 с.

6 **Павловская, Т. А.** C/C++. Программирование на языке высокого уровня : учебник для вузов / Т. А. Павловская. – СПб. : Питер, 2010. – с. 467.

7 **Павловская, Т. А.** C/C++. Структурное и объектно-ориентированное программирование : практикум / Т. А. Павловская, Ю. А. Щупак. – СПб. : Питер, 2010. – 329 с.

8 **Пахомов, Б. И.** C/C++ и MS Visual C++ 2010 для начинающих. – СПб. : БХВ-Петербург, 2011. – 736 с. : ил. + дистрибутив (на DVD).

9 **Понамарев, В. А.** Программирование на C++/C# в Visual Studio. NET / В. А. Пономарев. – СПб. : БХВ-Петербург, 2004. – 562 с.

10 **Хортон, Айвор** Visual C++ 2005: базовый курс : пер. с англ. / Хортон Айвор. – М. : ООО «И. Д. Вильямс», 2007. – 1152 с. : ил. – Парал. тит. англ.

ПРИЛОЖЕНИЕ А

График сдачи модулей
для студентов дневной формы обучения

(1-й триместр)

№ модуля	Дисциплина	Время на усвоение	Кредиты ECTS	Недели, формы контроля														
				1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	Вычислительная техника и программирование	18	0,5	1	1													
2	Вычислительная техника и программирование	72	2,0			2						2						
3	Вычислительная техника и программирование	54	1,5										3					3

II триместр

№ модуля	Дисциплина	Время на усвоение	Кредиты ECTS	Недели, формы контроля								
				1	2	3	4	5	6	7	8	9
4	Вычислительная техника и программирование	108	3,0	4								4

Приложение Б

Состав модулей дисциплины

«***Вычислительная техника и программирование***»

для студентов специальности **8.05070204 «Электромеханические системы автоматизации и электропривод»**

дневной формы обучения,

распределение времени на их усвоение, термины контроля

(1-й триместр)

Таблица Б.1 – Состав модулей дисциплины

Модуль	Краткое содержание модуля	Триместр	Общее количество часов	Кредиты ECTS	Количество аудиторных часов	Из них лабораторные работы	Формы и методы контроля	Количество баллов		Неделя проведения
								Макс	Мин	
1	**Основы работы с операционной системой Windows**	I				2	**Защита л. р. по Windows**. Работа с окнами. Управление файловой системой. Архивация	35	20	1
						3	**Защита л. р. по Word.** Ввод и форматирование текста. Работа с таблицами, редактор формул. Вставка объектов в документ	35	20	2
						1	**Защита РГР**	30	15	2
			18	0,5	8	**6**	**Всего за модуль**	**100**	**55**	**2**
2	**Основные конструкции языка программирование С++**	I				1	**Защита л. р. 1**. Знакомство со средой Visual Stidio. Создание простой программы на языке С++ в консоли	5	2	3
						1	**Защита л. р. 2**. Переменные и базовые типы данных языка С++. Создание программы линейного алгоритма	6	3	3
						2	**Защита л. р. 3**. Принятие решений. Условные операторы на языке С++	7	4	3-4
						2	**Защита л. р. 4**. Организация циклов на языке С++	10	5	4
						4	**Защита л. р. 5**. Одномерные числовые массивы на языке программирования С++. Селективная обработка элементов массива. Нахождение минимального и максимального элементов массива	16	9	5-6
						5	**Защита л. р. 6.** Понятие многомерного массива. Обработка элементов матриц	18	11	6-7

Продолжение таблицы Б.1

						2	**Защита л. р. 7.** Построение графика функции	14	8	8
						2	**Защита л. р. 8.** Файловый ввод и вывод на языке С++	12	6	8-9
						1	**Самостоятельная работа.** Обработка элементов диагоналей квадратных матриц	7	4	
						1	**Контрольная работа**	5	3	9
			72	2	29	**21**	**Всего за модуль**	**100**	**55**	**9**
3	**Основы программирования в среде Visual Studio 2010**	I				6	**Защита л. р. 1**. Знакомство с визуальной средой программирования Visual Studio 2010. Создания самого простого дополнения Windows	30	17	10
						6	**Защита л. р. 2.** Условные операторы. Вычисление значения функции, заданной условно	30	16	12
						5	**Защита л. р. 3.** Циклический алгоритм. Табулирование функции и поиск экстремумов	30	16	14
						1	**Контрольная работа**	10	6	15
			54	1,5	23	**18**	**Всего за модуль**	**100**	**55**	**15**
Всего за триместр			**144**	**4,0**	**60**	**45**	Весовые коэф. модулей: 0,1; 0,5; 0,4; *(Для триместра на модульный контроль)*			
4	**Разработка дополнений в среде Visual Studio 2010**	**II**				4	**Защита л. р. 4.** Построение графика функции на промежутке с определенным шагом.	11	6	1-2
						5	**Защита л. р. 5.** Понятие одномерного массива. Селективная обработка элементов массива	18	10	2-3
						6	**Защита л. р. 6.** Понятие двумерного массива (матрицы). Селективная обработка элементов строк, столбцов и диагоналей матрицы	19	10	4-5
						6	**Защита л. р. 7.** Изучение вероятностных алгоритмов (метод Монте-Карло)	15	8	6-7
						3	**Защита л. р. 8.** Робота с диалоговыми окнами. Создание операционного меню	11	6	8
						2	**Самостоятельная работа.** Обработка матрицы. Формирование одномерных массивов из двумерных. Сортировка одномерных массивов	16	9	9
						1	**Контрольная работа**	10	6	9
			108	**3,0**	**45**	**27**	**Всего за модуль**	**100**	**55**	
Всего по курсу			**252**	**7,0**	**105**	**72**	**Весовые коэфициенты за курс:** **М1 – 0,1** **М2 – 0,3** **М3 – 0,25** **М4 – 0,35**			

Навчальне видання

ЗАГРЕБЕЛЬНИЙ Сергій Леонідович,
КОСТІКОВ Олександр Анатолійович,
МІРІНСЬКИЙ Вадим Едуардович

ОСНОВИ ВІЗУАЛЬНОГО ПРОГРАМУВАННЯ У СЕРЕДОВИЩІ VISUAL STUDIO 2010

Навчальний посібник

для студентів спеціальності
8.05070204 «Електромеханічні системи
автоматизації та електропривод»

(Російською мовою)

Редагування С. П. Шнурік

Комп’ютерне верстання О. П. Ордіна

113/2012. Формат 60 х 84/16. Ум. друк. арк. 9,3.
Обл.-вид. арк. 8,65. Тираж 28 пр. Зам. № 131

Видавець і виготівник
Донбаська державна машинобудівна академія
84313, м. Краматорськ, вул. Шкадінова, 72.
Свідоцтво суб’єкта видавничої справи
ДК №1633 від 24.12.2003