**ЛАБОРАТОРНАЯ РАБОТА № 2.**

**Тема. Основы объектно-ориентированного программирования. Система визуального программирования Borland Delphi. Создание интерфейса программы. Визуальные компоненты для ввода и вывода данных.**

Основные вопросы:

* основные типы данных в Object Pascal.;
* компоненты для ввода и вывода данных;
* надпись Label, поле Edit;
* кнопка Button.;
* счѐтчик UpDown;
* поле Memo;
* основные свойства компонентов;
* вычисления в программе;
* группировка компонентов с использованием фреймов.

**Теоретическая часть**.

**Основные типы данных в Object Pascal.**

При создании любой программы (кроме самой элементарной) используются переменные величины. Любая используемая в программе переменная относится к определѐнному типу данных. Имя и тип переменной всегда задаѐтся до начала еѐ использования в программе в блоке объявления переменных.

В Delphi используется множество различных типов переменных. Все типы переменных можно разделить на шесть основных подразделов:

* простой тип;
* строковый тип;
* структурный (структурированный) тип;
* указательный тип;
* процедурный тип;
* вариантный тип.

**Простые типы**

Простые типы определяют упорядоченные множества значений.

**Целочисленные типы**

|  |  |  |
| --- | --- | --- |
| **Тип** | **Диапазон** | **Формат** |
| Integer | –2147483648..2147483647 | Знаковый 32–бит |
| Cardinal | 0..4294967295 | Беззнаковый 32–бит |
| Shortint | –128..127 | Знаковый 8–бит |
| Smallint | –32768..32767 | Знаковый 16–бит |
| Longint | –2147483648..2147483647 | Знаковый 32–бит |
| Byte | –2 | Знаковый 64–бит |
| Word | 63 | Беззнаковый 8–бит |
| Longword | ..2 | Беззнаковый 16–бит |
| Тип | 63 | Беззнаковый 32–бит |

**Вещественный тип**

|  |  |  |  |
| --- | --- | --- | --- |
| **Тип** | **Диапазон** | **Значащие цифры** | **Размер в байтах** |
| Real | 5.0\*10-324..1.7\*10308 | 15–16 | 8 |
| Real48 | 2.9\*10–39..1.7\*1038 | 11–12 | 6 |
| Single | 1.5\*10–45..3.4\*1038 | 7–8 | 4 |
| Double | 5.0\*10–324..1.7\*10308 | 15–16 | 8 |
| Extended | 3.6\*10–4951..1.1\*104932 | 19–20 | 10 |
| Comp | –263+1.. 263–1 | 19–20 | 8 |
| Currency | –922 337 203 685 477.5808 .. 922 337 203 685 477.5808 | 19–20 | 8 |

**Булев тип – Boolean**

Значением переменной булева типа может быть только одно из двух значений – False и True. Переменные булева типа иначе называются логическими. Значение True соответствует логической единице (Да, Правда), а значение False соответствуют логическому нулю (Нет, Ложь).

**Символьный тип – Char**

Значениями символьного типа является множество всех символов клавиатуры компьютера. Каждому символу приписывается целое число в диапазоне 0..255. Это число служит кодом внутреннего представления символа. Для кодировки в Windows используется код ANSI (American National Standard Institute – американский институт стандартизации, предложивший этот код).

**Перечислимый тип.**

Перечислимые типы определяют упорядоченные множества значений через перечисление идентификаторов, которые обозначают эти значения. Все идентификаторы в списке перечислимого типа описываются как константы. Типом этой константы является описанный перечислением тип.

Например, перечислимый тип для всех цветов радуги будет описан так:

**type rainbow = (red, orange, yellow, green, cyan, blue, purple);**

**Строковый тип – String**

Значением строковой переменной является последовательность символов с динамически изменяемой длиной. Для строковых переменных допускается использование пустой строки. Значение строковой переменной всегда заключается в символ одиночной кавычки – апостроф. Например, S := ‘Новороссийск’.

К элементам строки, т.е. отдельным символам, можно обращаться как к элементам массива. Например, для переменной S значением S[5] будет буква ‘р’ – пятая по счѐту в данной строке. Для пустой строки значение задаѐтся как пара апострофов без пробела внутри. S := ‘’;

**Структурные типы**

***Тип «Массив»***

Массивы содержат фиксированное число элементов одного типа. Для доступа к элементам массива используются индексы. Массив может иметь неограниченное число размерностей. Все элементы массива имеют один и тот же тип элементов – базовый тип. Базовым типом может быть любой из порядковых типов, диапазон которых не превышает 2 Гбайта.

Например, для объявления одномерного массива **S**, состоящего из 10 элементов целого типа:

**var**

**S: array [1..10] of integer;**

Двумерный массив целых чисел **S1** размерностью 10х10 может быть объявлен одним из следующих способов:

**var**

**S1: array [1..10] of array [1..10] of integer;**

или в более компактной форме:

**var**

**S1: array [1..10,1..10] of integer;**

**Тип «Запись»**

Тип «запись» содержит установленное число элементов или полей, которые могут быть разных типов. В описании записи указывается идентификатор, который именует поле, и тип каждого поля. Например, описание нового типа элементов, который является записью:

**type**

**Student = record**

**Name: string;**

**Family: string;**

**Year: integer;**

**end;**

**Тип «Класс»**

Тип «класс» является структурой, состоящей из фиксированного числа компонентов. Каждый компонент класса является либо полем, содержащим данные строго определѐнного типа, либо методом (процедурой или функцией). В Delphi используются огромное число стандартных классов, каждый из которых содержит разнообразные методы. Программист может и сам разрабатывать новые нестандартные классы, определять поля, методы для решения прикладных задач. Методика разработки и использования в программах нестандартных классах – тема отдельного разговора.

**Тип «Множество»**

Это математический термин, и он применим в данном случае, так как этот тип представляет именно множество, и над ним можно производить все те же действия, что и над множествами. Диапазон значений множества представляет собой мощность множества для определѐнного порядкового типа (базового типа).

Каждое возможное значение множества является подмножеством возможных значений базового типа. В *Pascal* количество значений базового типа не должно превышать 256, и порядковые значения верхней и нижней границы базового типа не должны превышать диапазона от 0 до 255. В силу этого базовый тип множества не может быть коротким целым (*Shortint*), целым (*Integer*), длинным целым (*Longint*) или словом (*Word*).

Переменная для множества может принимать как все значения множества, так и ни одного. Множество можно создать так:

**type**

**TIntSet = set of 1 .. 250;**

**Тип «Файл»**

Использование файловых переменных является одной из важнейших задач при создании прикладных программ и технология их использования будет рассмотрена позднее.

**Создание интерфейса программы.**

Интерфейс программы - совокупность элементов, позволяющих пользователю программы управлять еѐ работой, а также получать требуемые результаты. Фактически, пользовательский интерфейс - это канал, по которому осуществляется взаимодействие пользователя и программы. Почему есть необходимость вообще говорить о программном интерфейсе?

Дело в том, что исходя из самой идеи, программа пишется для пользователя, для удовлетворения его потребностей. В итоге же часто получается, что программист пишет программу «для себя», т.е. никому больше она не нужна, потому что кроме автора этой программы понять ее мало кто смог.

И можно, конечно, доказывать, что алгоритм программы гениальный, и, что код работает

очень быстро, занимает меньше места по сравнению с аналогичными программами, но эти моменты будут интересовать только программистов, потому что будут понятны только им и только ими будут оценены должным образом. Пользователь же предъявляет совершенно иные требования, чем программист, так как смотрит на код снаружи, а не изнутри. И, соответственно, большей популярностью будет пользоваться та программа, прежде всего, с которой пользователи нашли общий язык, остальные же параметры интересуют пользователей во вторую очередь.

В основе работы любой диалоговой программы лежит взаимодействие пользователя и программы через еѐ интерфейс. При этом пользователь программы различными способами вводит или выбирает данные, а программа обрабатывает эти данные и возвращает результат в различных видах – текстовом, табличном, графическом и других.

Для этих целей в любой современной визуальной системе программирования есть множество компонент для размещения и использования в интерфейсной части программы. Рассмотрим несколько основных компонентов: текстовую надпись *Label*, поле для ввода Edit, кнопка Button, списки *ListBox* и *ComboBox*.

**Текстовая надпись Label**

При оформлении интерфейса программы необходимы надписи различного вида. Для этого можно использовать объект *Label* – надпись.

**Свойства класса TLabel**

|  |  |
| --- | --- |
| **Свойство** | **Назначение** |
| Align | Указывает выравнивание надписи относительно родительского объекта –  формы. По умолчанию – alNone |
| Alignment | Выравнивание текста надписи относительно границ компонента по  горизонтали. |
| Autosize | Автоматическое изменение размера компонента в зависимости от размера  текста в свойстве Caption с учѐтом установленного шрифта. Если свойство  имеет значение False, высота и ширина компонента определяется текущими  значениями свойств Height и Width. |
| Caption | Текст надписи |
| Color | Определяет цвет фона компонента |
| Cursor | Определяет вид указателя мыши при наведении еѐ на компонент. |
| Enabled | Активность компонента. Если компонент имеет значением свойства False, то  цвет текста надписи будет серым – неактивным. |
| Font | Задаѐт параметры шрифта компонента |
| Height | Высота компонента в пикселях |
| Hint | Текст всплывающей подсказки при наведении указателя мыши |
| Left | Положение компонента от левой границы формы в пикселях |
| Name | Имя компонента для обращения к нему из программы. |
| ShowHint | Подсказка из свойства Hint может быть отображена только при значении  свойства True |
| Top | Положение компонента от верхней границы формы в пикселях |
| Transparent | Определяет прозрачность метки: если содержит значение False,  пространство метки закрашивается собственным цветом Color, в противном  случае этого не происходит и сквозь метку будут видны расположенные за  ней компоненты (так, например, можно наложить текст на графику) |
| Visible | Определяет видимость компонента. Если содержит значение False, то  компонент остаѐтся, но становится невидимым при прогоне программы.  Таким способом можно делать одни компоненты видимыми, другие скрывать. |
| Width | Высота компонента в пикселях. |

**Однострочное поле ввода и редактирования – Edit.**

Компоненты класса *TEdit* представляют собой однострочное текстовое поле для редактирования. Основное свойство компонента *Text* – содержит отображаемую в поле строку. Это свойство используется для ввода и вывода данных в поля *Edit*.

**Очень важно!**

Для вычислений необходимо использовать переменные, относящиеся к числовому типу. Особенностью *Delphi* (и многих других визуальных систем программирования) является то, что данные, введѐнные в поля *Edit*,изначально относятся к символьным данным. Для выполнения математических действий с ними как с числами вначале необходимо преобразовать их из символьных в числовые, а затем выполнять действия с числовыми переменными. Вначале введѐнные значения в поля Edit необходимо присвоить строковым переменным, преобразовать их в числовое значение (целое или вещественное) и только потом выполнять с полученными числовыми переменными вычисления.

Аналогично, полученное числовое значение перед выводом в поле Edit необходимо преобразовать в символьное значение. Для этого используются функции присвоения и преобразования.

Для присвоения значения поля Edit1 символьной переменной S используется функция:

**S := Edit1.Text;**

Для преобразования символьного значения переменной S в целочисленное значение X используется функция:

**X := StrToInt(S);**

Для преобразования символьного значения переменной S в вещественное значение X используется функция:

**X := StrToFloat(S);**

Для обратного преобразования используются аналогичные функции:

**S: = IntToStr (X);**

**S: = FloatToStr (X);**

Для вывода значения символьной переменной S в поле Edit используется функция:

**Edit1.Text := S;**

**Многострочное поле ввода и редактирования – Memo.**

Используется в программах для вывода или ввода большого объѐма информации, чаще всего в несколько строк. Примером может быть ввод и отображение полного почтового адреса в адресных книгах.

Многие свойства данного компонента аналогичны соответствующим свойствам других компонентов. Главное свойство – *Lines* – хранит текст. Текст представляет собой пронумерованный набор строк (нумерация начинается с нуля). Содержимое поля может быть задано через Инспектор объектов (свойство *Lines*), введено при прогоне программы, сформировано динамически (данный способ будет рассмотрен при работе с циклами).

**Свойства компонента TMemo**

|  |  |
| --- | --- |
| **Свойство** | **Назначение** |
| CaretPos | Содержит координаты мигающего текстового курсора относительно границ  клиентской области компонента. |
| Lines | Содержит строки текста |
| ScrollBars | Определяет наличие в многострочном поле полос прокрутки:   * ssNone – нет полос; * ssHorizontal – есть горизонтальная; * ssVertical – есть вертикальная; * ssBoth – есть обе полосы. |
| Text | Отображает содержимое свойства Lines в виде одной длинной строки, в  которой границы отдельных строк определяются символами EOL (CR + LF) |
| WantReturns | Если содержит значение True, нажатие клавиши Enter вызывает переход  на новую строку, в противном случае – обрабатывается системой. Для  перехода в новую строку в этом случае следует нажать клавиши Ctrl + Enter |

Остальные свойства данного компонента будут рассмотрены позднее.

**Кнопка – Button**

Один из основных элементов управления в программах. Связанный с кнопкой алгоритм управления реализуется в процедуре *OnClick*.

Основные свойства этого компонента аналогичны соответствующим свойствам других компонентов. Надпись на кнопке создаѐтся с использованием свойства *Caption*.

Компонент *UpDown* – счѐтчик предназначен для пошагового регулирования числовой величины. Он имеет пару кнопок, с помощью которых величина наращивается или уменьшается. Сам по себе этот элемент не имеет возможности что–либо менять, а обычно его связывают с текстовым полем *Edit*, которое отображает регулируемую величину.

Связанный компонент называется компаньоном. Сам компонент *UpDown* находится во вкладке *Win32*.

***Свойства компонента UpDown***

|  |  |
| --- | --- |
| **Свойство** | **Назначение** |
| AlignButton | Определяет положение счѐтчика относительно компаньона:   * udLeft – счѐтчик располагается слева от него; * udRight – справа от него. |
| ArrowKeys | Разрешает/запрещает интерпретировать нажатие клавиш ↑ и ↓ как щелчки,  соответственно на направленной вверх и направленной вниз кнопках  счѐтчика. |
| Associate | Определяет связанный компонент (компаньон). Кнопки автоматически  располагаются с нужной стороны компаньона и выравнивают свою высоту  по его высоте. |
| Increment | Определяют шаг наращивания/уменьшения регулируемой величины.  Значение может иметь только целочисленное значение – SmallInt.  (–32768..32767) |
| Max и Min | Определяет максимальное и минимальное значение диапазона изменения  регулируемой величины. Значение может иметь только целочисленное  значение – SmallInt. |
| Orientation | Определяет ориентацию компонента:   * *udHorizontal* – по горизонтали; * *udVertical* – по вертикали. |
| Position | Содержит текущее значение текущей величины. Значение может иметь  только целочисленное значение – *SmallInt*. |
| Thousands | Если содержит значение *True*, при отображении числовой величины в  компаньоне в текст вставляются разделители тысяч |
| Wrap | Запрещает/разрешает выход значения Position из диапазона Max*…Min* (*True*  – запрещает). |

Для создания счѐтчика в окно проекта добавляется два компонента – *Edit* и *UpDown.* Для присоединения к компоненту *UpDown* поля *Edit* в свойствах первого необходимо открыть свойство *Associate* и выбрать из списка компонент–компаньон. Полученный объект и есть счѐтчик в том виде, как мы его привыкли видеть и использовать во всех приложениях *Windows*.

**Группировка компонентов с использованием фреймов.**

Фреймы (англ*. frame* – рамка) – это контейнер для объединения в одну логическую группу нескольких компонентов. В проектах с простым интерфейсом, где количество компонентов невелико, группировать компоненты нет необходимости. Однако в проектах, где используются компоненты, относящиеся к различным логическим группам, наиболее удобным средством размещения и управления свойствами компонентов являются фреймы. Примером использования фреймов во многих приложениях являются панели инструментов.

Добавление фрейма в проект происходит в два этапа:

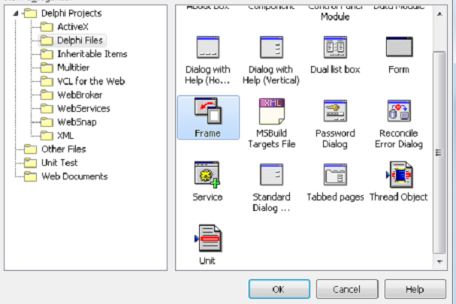
* меню *File* 🡺 *New* 🡺 *Other* 🡺 в диалоговом окне *New Items* в категории *Delphi Files* выбрать компонент Frame. При этом фрейм создаѐтся в проекте, но не отображается;
* окно (форму) проекта фрейм добавляется с использованием визуального компонента *Frame* в *Палитре компонентов* *Standard* (см. рис. 1).

Рис. 1

При первой попытке добавления из *Палитры компонентов* раздела *Standard* компонента *Frame* в форму, система *Delphi* выдаѐт сообщение (рис. 2) о том, что в проекте нет созданных фреймов и предлагается путь к созданию фреймов.

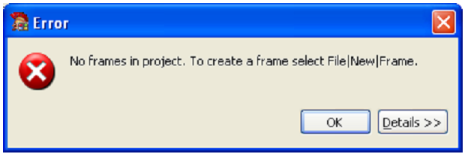


Рис. 2

В дальнейшем в проект можно добавлять аналогично несколько фреймов. Однако после создания фрейма с помощью меню его необходимо добавить в форму из Палитры компонентов *Standard*. Для каждого нового фрейма будет создан отдельный модуль (*unit*) и форма для добавления в него компонентов.

В этом случае фрейм является контейнером для нескольких компонентов, управление которыми выполняется не в отдельности, а всем фреймом в целом. Например, изменение положение фрейма. В этом случае все компоненты фрейма являются его дочерними компонентами.

**Простые и структурированные операторы Object Pascal**

Операторы *Object Pascal* подразделяются на две группы: *простые* и *структурированные*. Простые операторы не содержат в себе других операторов; структурированные включают в себя другие операторы – как простые, так и структурированные. К простым операторам относятся *оператор присваивания*, *оператор перехода* (***GOTO***) и *пустой оператор*, а к структурированным – *составной оператор* (***BEGIN…END***), *услов­ные операторы* (***IF*** и ***CASE***) и *операторы цикла* (***WHILE***, ***REPEAT*** и ***FOR***).

*Простые операторы.*

*Оператор присваивания* записывается в виде:

<*P*> **:**= <*W*>;

В состав оператора присваивания входит знак присваивания **:**=, который делит этот оператор на две части. Левая часть оператора <*P*> – содержит переменную или типизированную константу любого типа, а в правой части находится выражение <*W*> (состоящее из идентификаторов кон­стант, переменных, функций и знаков операций), которое необходимо вычислить. После вычисления полученное значение присваивается переменной, указанной в левой части оператора. При этом тип переменной должен быть совместим с ти­пом вычисленного выражения.

*Оператор передачи управления.*

*Оператор передачи управления* имеет вид:

***goto*** *mtk****,***

где *mtk* – метка, которой помечен оператор в теле программы, выполнение которого последует непосредственно после выполнения оператора ***goto.*** Другими словами оператор ***goto*** позволяет прервать последовательный ход выполнения операторов и передать управление в произвольную точку кода, помеченную специальной меткой. Метки, на которые может передаваться управление, объявляются ключевым словом ***Label***, после которого следует список меток. Все три элемента данной конструкции: объявление меток, сами метки и операторы ***goto,***  должны располагаться в рамках одного блока. Каждая метка может обозначаться допустимым идентификатором или числом от 0 до 9999. Например, следующее объявление

***Label*** *mtk1, mtk2, 3;*

объявляет три метки: *mtk1, mtk2* и *3*.

В тексте программы метка отмечает точку, в которую передается управление оператором ***goto***. Метка может располагаться в любом месте блока, как после оператора ***goto***, передающего на нее управление, так и до этого оператора. Надо только иметь в виду; что передача управления извне внутрь цикла может приводить к непредсказуемым последствиям, так что таких ситуаций следует избегать. Не разрешается также передавать управление на метку, расположенную в другом блоке.

Сама точка передачи управления, помечается меткой, после которой ставится двоеточие и следует оператор, на который передается управление. Например:

*Mtk1: Read (m*);

*Пустой оператор.*

*Пустой оператор* не выполняет никаких действий и в тексте программы о его присутствии свидетельствует только один символ «точка с запятой». Нам уже известно, что в конце составного оператора, перед ключевым словом ***end,*** точка с запятой необязательна. Однако, в случае, когда рядом со словом ***end*** расположена метка, перед меткой обязательно должна присутствовать точка с запятой, назначение которой в данном случае – послужить разделителем между последним реальным оператором и пустым оператором.

*Структурированные операторы.*

*Составной оператор.*

*Составной оператор* представляет собой последовательность некоторых операторов, которые выполняются в том порядке, в каком они содержатся в тексте программы. При этом ключевые слова ***BEGIN*** и ***END*** являются так называемыми *операторными скобками, в* которые заключены операторы, вхо­дящие в составной оператор. Когда, в зависимости от определенного условия, требуется обеспечить последовательное выполнение некоторого набора опера­торов, без составного оператора не обойтись. Составной оператор имеет следующий вид:

***begin*** *P1; P2; … ; Pn*  ***end;***

Здесь *P1 – Pn –* операторы, образующие составной оператор; ключевые слова ***BEGIN*** и ***END*** *–* операторные скобки. Составной оператор может включать другие составные операторы (т.е. составным может быть любой из операторов *P1 – Pn,* причем допускается любой уровень вложенности).

**Практическая часть**

**Ход работы.**

**Задание 1.**

**Создание интерфейса программы.**

1. Создать новый проект.

2. Для окна проекта в Инспекторе объектов задать следующие свойства:

* Размер формы: ширина – 600 пикселей, высота – 600 пикселей.
* Заголовок формы – Лабораторная работа № 2.

3. В окно проекта из *Палитры компонентов*, вкладки *Standard* добавить текстовую надпись *Label1* в свойствах которой настроить:

* *Caption* – ввести свои *фамилию, имя, отчество и номер варианта*.
* *Font*  *Size* (*Размер*) задать 14 размер.
* Поместить данный заголовок в верхней части формы.

4. Аналогично добавить ещѐ одну текстовую надпись *Label2*, в свойство *Caption* ввести *Задание1. Вычисление стоимости товара.* Свойства шрифта и размещение объекта в форме задать произвольно.

5. В форму добавить ещѐ три надписи с текстом: *Количество*, *Цена*, *Стоимость*.

6. Добавить три текстовых поля *Edit* для дальнейшего их использования: для ввода числовых значений количества, цены товара и вывода стоимости. Разместить их под соответствующими заголовками. Для этих полей свойство *Text* сделать пустым. Значение данного свойства – это тот текст, который автоматически отображается при прогоне программы. Размер каждого из полей задать графически.

7. Добавить в форму две кнопки – объект *Button*. Для одной из них в свойство *Caption* ввести *Расчѐт*, для другой – *Выход из программы*. Размеры и положение кнопок задать произвольно.

**Создание кода программы.**

1. Логика работы программы такова, что после ввода числовых значений в поля *Количество* и *Цена* при щелчке по кнопке *Расчѐт* программа должна рассчитать стоимость товара по формуле **[Стоимость] = [Количество] \* [Цена]** и полученное значение вывести в поле *Стоимость*.

2. Для выполнения всех этих действий необходимо для кнопки *Расчѐт* (*Button1*) создать процедуру по событию *OnClick*. Для этого выделить объект *Button1* и в *Инспекторе объектов* во вкладке *Events* (*События*) дважды щѐлкнуть в строке события *OnClick*.

Будет создана процедура следующего вида:

**procedure TForm1.Button1Click(Sender: TObject);**

**begin**

**end;**

3. Для передачи в программу введѐнных в поля Количество и Цена данных, а затем для вывода в поле Стоимость необходимо использовать в процедуре три переменных строкового типа. Например, KolS, CenaS, StoimS – символ S в конце имени каждой переменной условно обозначает, что данные переменные относятся к строковому типу. Для преобразования этих строковых переменных в числовые необходимо использовать в программе ещѐ три переменных числового формата.

Для определѐнности выберем целочисленные. (Вычисления будем выполнять с целым количеством и ценой, представленной целым числом.). Переменные KolInt, CenaInt, StoimInt – целочисленные.

4. Переменные должны быть объявлены после заголовка процедуры перед телом процедуры. Например,

**procedure TForm1.Button1Click(Sender: TObject);**

**var**

//Строковые переменные для количества, цены и стоимости

**KolS, CenaS, StoimS: string;**

//Целочисленные переменные для количества, цены и стоимости

**KolInt, CenaInt, StoimInt: integer;**

5. В основной части процедуры необходимо последовательно из полей с ценой и количеством введѐнные значения присвоить соответствующим символьным переменным, преобразовать в числа, выполнит вычисления и преобразовать результат в символьное значение. Затем его вывести в соответствующее поле.

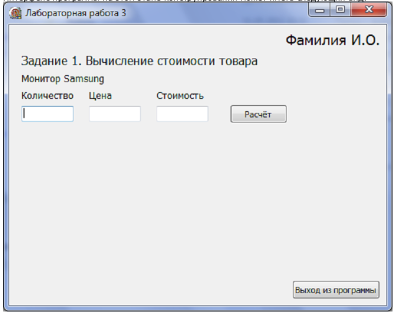
 Интерфейс программы на этом этапе конструирования может иметь следующий вид – рис.3

Рис. 3

**begin**

//Считываем из поля Количество введённые данные как строковое значение

**KolS := Edit1.Text;**

//Считываем из поля Цена введённые данные как строковое значение

**CenaS := Edit2.Text;**

Затем полученные значения преобразовать в числовые значения.

//Преобразуем символьные значения в целочисленные

**KolInt := StrToInt(KolS);**

**CenaInt := StrToInt(CenaS);**

После преобразования можно вычислять значение стоимости товара в виде целого числа.

//Вычисляем стоимость по формуле

**StoimInt := KolInt \* CenaInt;**

Но перед выводом полученного числового значения в поле Стоимость его необходимо преобразовать в символьное:

//Полученное числовое значение стоимости преобразуем в символьное

**StoimS := IntToStr(StoimInt);**

Для вывода символьного значения в поле Стоимости его значение необходимо присвоить свойству Text соответствующего поля.

//Символьное значение выводим в поле Стоимость

**Edit3.Text := StoimS;**

**end;**

6. Для выхода из программы можно использовать стандартные средства окна проекта, а можно для этого запрограммировать соответствующую кнопку Выход. Для этой кнопки необходимо создать процедуру на событие OnClick, а код процедуры будет содержать только одну единственную команду – метод формы *Close*.

7. Для проверки работоспособности программы сохраните проект под именем *Lab2* в отдельную папку и запустите программу на выполнение. Убедитесь, что программа корректно работает в случае ввода в поля *Количество* и *Цена* целых чисел.

**Задание 2. Интерфейс**

1. В окно созданного проекта добавить текстовый заголовок Задание 2. Управление интерфейсом.

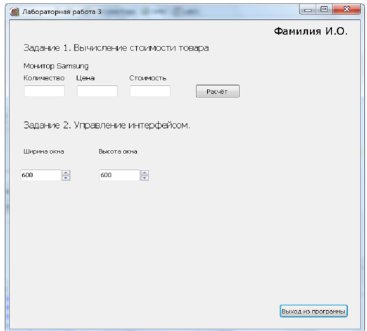
2. Добавить два текстовых однострочных поля *Edit* и два компонента *UpDown*. В свойствах компонентов *UpDown* указать, с каким текстовым полем будет работать этот компонент. Кроме того, для ограничения диапазона ввода в счѐтчиках в свойствах компонентов *UpDown* задать минимальное значение (*Min*) равное 600 и максимальное значение (*Max*) равное 800. На данном этапе конструирования интерфейс приложения может иметь вид – рис. 4.

Рис. 4

Основное свойство компонента *UpDown* – *Associate*, которое связывает кнопку *UpDown* (кнопка со стрелками) с одним из оконных компонентов, обычно с *Edit*. Чтобы опробовать компонент *UpDown*, перенесите на форму его и окно редактирования *Edit,* расположив *Edit* там, где это требуется, а *UpDown* — в любом месте формы. Далее в выпадающем списке свойства *Associate*компонента *UpDown* выберите *Edit1*. Компонент *UpDown* немедленно переместится к *Edit* и как бы сольется с ним.

**Программный код**

3. Логика работы заключается в том, что при использовании счѐтчика ширины и высоты формы в компоненте *Edit* происходит изменение значения. В зависимости от заданного значения размеры окна должны меняться. Для написания программного кода в каждом из полей *Edit*, привязанных к счѐтчику, необходимо использовать событие *onChange*. С этой целью выделяем на форме необходимый компонент (*Edit4* или *Edit5*) и «кликнув» по нему два раза, попадаем на страничку с программным кодом.

Например, для поля *Edit4* со значением ширины окна процедура может иметь следующий вид:

**procedure TForm1.Edit4Change(Sender: TObject);**

**begin**

**form1.Width := StrToInt(Edit4.Text);**

**end;**

При этом ввести программный код необходимо только в 3–й строке этой процедуры. Создать процедуру изменения высоты окна предлагается ввести самостоятельно по аналогии с предыдущим примером. Здесь функция *StrToInt* (), обеспечивает перевод строковой переменной (*String*) в целочисленный тип (*Integer*).

**Задание 3. Интерфейс.**

1. Добавить в проект фрейм и вставить его в форму (окно) проекта. При добавлении визуального компонента *Frame* из *Палитры компонентов* автоматически создаѐтся новый модуль и форма для него. В форму созданного фрейма (она отобразится в отдельной вкладке) добавить компоненты: текстовая надпись *Label*, однострочное поле для ввода и редактирования *Edit*, кнопку *Button* и многострочное поле *M*e*mo*. В компонентах *Label* и *Button* задать свойство *Caption*. В поле *Edit* в свойстве *Text* удалить всѐ содержимое. В поле *Memo* для удаления стандартного текста в *Инспекторе объектов* необходимо открыть свойство *Lines*. Именно это свойство позволяет управлять содержимым поля *Memo* и с помощью *Инспектора объектов* и программным путѐм.

На данном этапе проектирования интерфейс фрейма может иметь следующий вид – рис. 5.

2. *Frames* – это панель, некий фрагмент окна приложения, способный использоваться на разных формах, в разных приложениях и допускающий использование наследования. Прежде чем использовать компонент *Frames*, необходимо создать сам фрейм через меню ‘*File/New/ other*’ в открывающемся окне выбираем ‘*Delphi projects/delphi files*’ справа находим ‘*Frame*’ (см. рис 6), кликнув на по нему двойным щелчком создаем фрейм – знакомое уже нам окно формы, но немного c другими свойствами. На нем так же, как и на форме, вы можете размещать любые компоненты, менять их свойства, создавать обработчики событий и т.д.

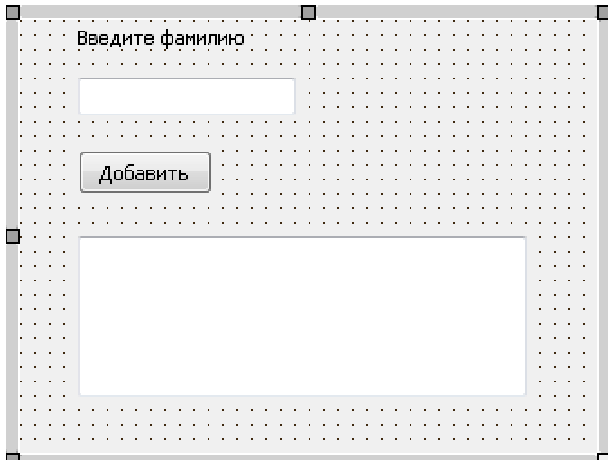


Рис. 5

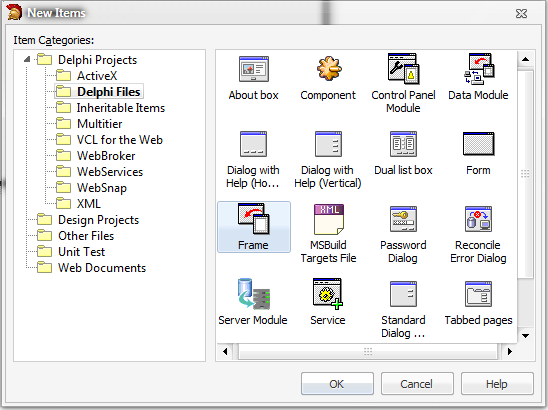


Рис 6

Однако не забываем, что пока нами создан всего лишь сам фрейм и он ни к чему не привязан. Далее нам как раз и понадобится компонент *frame* с вкладки *Standart*, который необходимо перетащить на форму нашего основного проекта. Сразу после добавления фрейма на форму появляется окно выбора ‘*Select frame to insert’*, из списка ниже выбираем нужный фрейм(см рис 7).

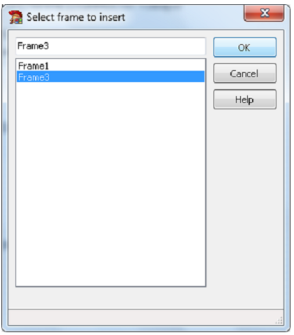


Рис. 7

Интерфейс программы на этом этапе работы может иметь следующий вид (рис. 8):

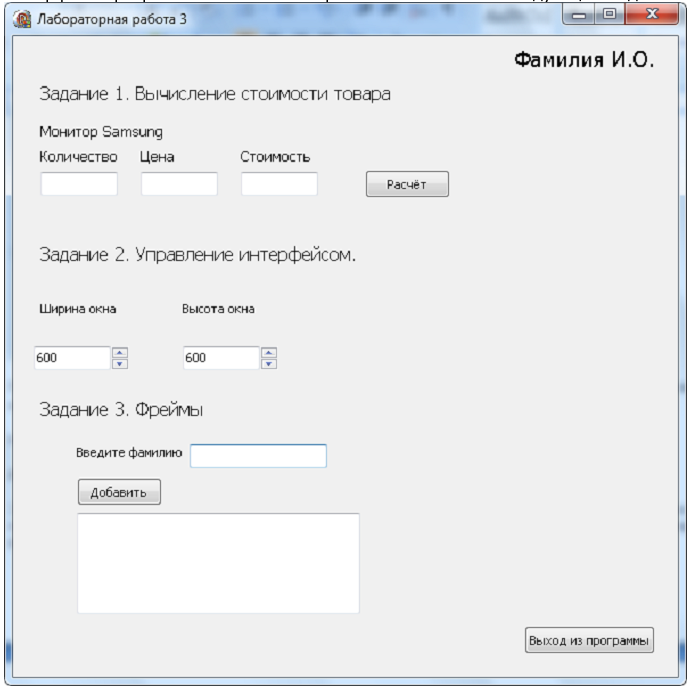


Рис. 8

**Программный код**

3. Логика работы программы должна быть такова, что после ввода в поле *Edit* требуемых данных, пользователь должен нажать кнопку *Добавить* после чего введѐнный текст добавляется в многострочное поле *Memo*, а содержимое поля *Edit* автоматически очищается.

Для кнопки *Button* необходимо создать процедуру по событию *onClick*.

В этой процедуре введены в поле *Edit* текст и преобразовывать в другой вид данных нет необходимости, поэтому можно в разделе переменных объявить имя одной строковой переменной. Раздел переменных может иметь следующий вид:

**var s: string;**

В теле процедуры программный код может иметь следующий вид:

//процедура обработки текстовых данных.

**s:=Frame21.Edit1.Text;**

**Frame21.Memo1.Lines.Add(s);**

**Frame21.Edit1.Text := '';**

Переменной s присваивается значение текстового поля *Edit*, но так как это поле вставлено не в форму непосредственно, в фрейм, то вначале перед именем поля Edit1 указано имя фрейма *Frame31* (это имя фрейма в данном примере!).

Для вывода текста в поле *Memo1* используется его свойство *Lines*, а для него ещѐ и метод *Add*. Для очистки поля *Edit1* после добавления текста в многострочное поле используется пустое значение. И в каждом случае перед именем компонента необходимо указать имя его родительского контейнера *Frame31*.

**Задание 4.**

***Вариант 1.***

*1.1 Разработать пользовательский интерфейс с использованием компонентов TEdit, TButton и TLabel, с помощью которых осуществляется вычисление арифметических выражений по формулам: Y1 = (X1-Х2)2 – (X3 – Х2)3, где X1, X2, X3 – значения вещественных чисел, введенных в три текстовых поля. Вставить в нижнюю часть формы Вашего (окна) проекта фрейм, разработанный в задании 3. Аналитический вид формулы поместить в интерфейс.*

***Вариант 2.***

*2.1 Разработать пользовательский интерфейс с использованием компонентов TEdit, TButton и TLabel, с помощью которых осуществляется вычисление арифметических выражений по формулам: Y1 = (X1 – Х2)2 \* (X3 + Х2)3, где X1, X2, X3 – значения вещественных чисел, введенных в три текстовых поля. Вставить в верхнюю часть формы Вашего (окна) проекта фрейм, разработанный в задании 3. Аналитический вид формулы поместить в интерфейс.*

***Вариант 3.***

*3.1 Разработать пользовательский интерфейс с использованием компонентов TEdit, TButton и TLabel, с помощью которых осуществляется вычисление арифметических выражений по формулам: Y1 = (X1 – Х2)4 – (X3/5 + Х2)3, где X1, X2, X3 – значения вещественных чисел, введенных в три текстовых поля. Вставить в левую часть формы Вашего (окна) проекта фрейм, разработанный в задании 3. Аналитический вид формулы поместить в интерфейс.*

***Вариант 4.***

*4.1 Разработать пользовательский интерфейс с использованием компонентов TEdit, TButton и TLabel, с помощью которых осуществляется вычисление арифметических выражений по формулам: Y1 = (X1/33 – Х22 – (X3 + Х2)2, где X1, X2, X3 – значения вещественных чисел, введенных в три текстовых поля. Вставить в правую часть формы Вашего (окна) проекта фрейм, разработанный в задании 3. Аналитический вид формулы поместить в интерфейс.*

***Вариант 5.***

*5.1 Разработать пользовательский интерфейс с использованием компонентов TEdit, TButton и TLabel, с помощью которых осуществляется вычисление арифметических выражений по формулам: Y1 = (X1 – Х2)/25 + (X3 – Х2)3, где X1, X2, X3 – значения вещественных чисел, введенных в три текстовых поля. Вставить в нижнюю часть формы Вашего (окна) проекта фрейм, разработанный в задании 3. Аналитический вид формулы поместить в интерфейс.*

***Вариант 6.***

*6.1 Разработать пользовательский интерфейс с использованием компонентов TEdit, TButton и TLabel, с помощью которых осуществляется вычисление арифметических выражений по формулам: Y1 = (X1 \* Х2)2 – (X3 – Х2/2)2, где X1, X2, X3 – значения вещественных чисел, введенных в три текстовых поля. Вставить в верхнюю часть формы Вашего (окна) проекта фрейм, разработанный в задании 3. Аналитический вид формулы поместить в интерфейс.*

***Вариант 7.***

*7.1 Разработать пользовательский интерфейс с использованием компонентов TEdit, TButton и TLabel, с помощью которых осуществляется вычисление арифметических выражений по формулам: Y1 = (X1 – 25\*Х2)2 – (X3 – Х2)4, где X1, X2, X3 – значения вещественных чисел, введенных в три текстовых поля. Вставить в боковую часть формы Вашего (окна) проекта фрейм, разработанный в задании 3. Аналитический вид формулы поместить в интерфейс.*

***Вариант 8.***

*8.1 Разработать пользовательский интерфейс с использованием компонентов TEdit, TButton и TLabel, с помощью которых осуществляется вычисление арифметических выражений по формулам: Y1 = (X1 – Х2/4)3 – (X3/3 – Х2), где X1, X2, X3 – значения вещественных чисел, введенных в три текстовых поля. Вставить в левую часть формы Вашего (окна) проекта фрейм, разработанный в задании 3. Аналитический вид формулы поместить в интерфейс.*

***Вариант 10***

*10.1 Разработать пользовательский интерфейс с использованием компонентов TEdit, TButton и TLabel, с помощью которых осуществляется вычисление арифметических выражений по формулам: Y1 = (X1 – Х2/7)2 – (X3 \* Х2/5), где X1, X2, X3 – значения вещественных чисел, введенных в три текстовых поля. Вставить в правую часть формы Вашего (окна) проекта фрейм, разработанный в задании 3. Аналитический вид формулы поместить в интерфейс.*

***Вариант 11.***

*11.1 Разработать пользовательский интерфейс с использованием компонентов TEdit, TButton и TLabel, с помощью которых осуществляется вычисление арифметических выражений по формулам: Y1 = (X1/100+Х2)2 – (X3 – Х2/50)2, где X1, X2, X3 – значения вещественных чисел, введенных в три текстовых поля. Вставить в верхнюю часть формы Вашего (окна) проекта фрейм, разработанный в задании 3. Аналитический вид формулы поместить в интерфейс.*

***Вариант 12.***

*12.1 Разработать пользовательский интерфейс с использованием компонентов TEdit, TButton и TLabel, с помощью которых осуществляется вычисление арифметических выражений по формулам: Y1 = (X1/3 – Х2)2 \* (X3 – Х2/5)3, где X1, X2, X3 – значения вещественных чисел, введенных в три текстовых поля. Вставить в нижнюю часть формы Вашего (окна) проекта фрейм, разработанный в задании 3. Аналитический вид формулы поместить в интерфейс.*

***Вариант 13.***

*13.1 Разработать пользовательский интерфейс с использованием компонентов TEdit, TButton и TLabel, с помощью которых осуществляется вычисление арифметических выражений по формулам: Y1 = (X1 – Х2)2 + (15\*X3 – Х2)3, где X1, X2, X3 – значения вещественных чисел, введенных в три текстовых поля. Вставить в боковую часть формы Вашего (окна) проекта фрейм, разработанный в задании 3. Аналитический вид формулы поместить в интерфейс.*

***Вариант14.***

*14.1 Разработать пользовательский интерфейс с использованием компонентов TEdit, TButton и TLabel, с помощью которых осуществляется вычисление арифметических выражений по формулам: Y1 = (5\*X1 – Х2)/9 – (X3 + Х2)/7, где X1, X2, X3 – значения вещественных чисел, введенных в три текстовых поля. Вставить в левую часть формы Вашего (окна) проекта фрейм, разработанный в задании 3. Аналитический вид формулы поместить в интерфейс.*

***Вариант 15.***

*15.1 Разработать пользовательский интерфейс с использованием компонентов TEdit, TButton и TLabel, с помощью которых осуществляется вычисление арифметических выражений по формулам: Y1 = (X1 – Х2)2 + (X3/6 – 17\*Х2)3, где X1, X2, X3 – значения вещественных чисел, введенных в три текстовых поля. Вставить в правую часть формы Вашего (окна) проекта фрейм, разработанный в задании 3. Аналитический вид формулы поместить в интерфейс.*

***Вариант16.***

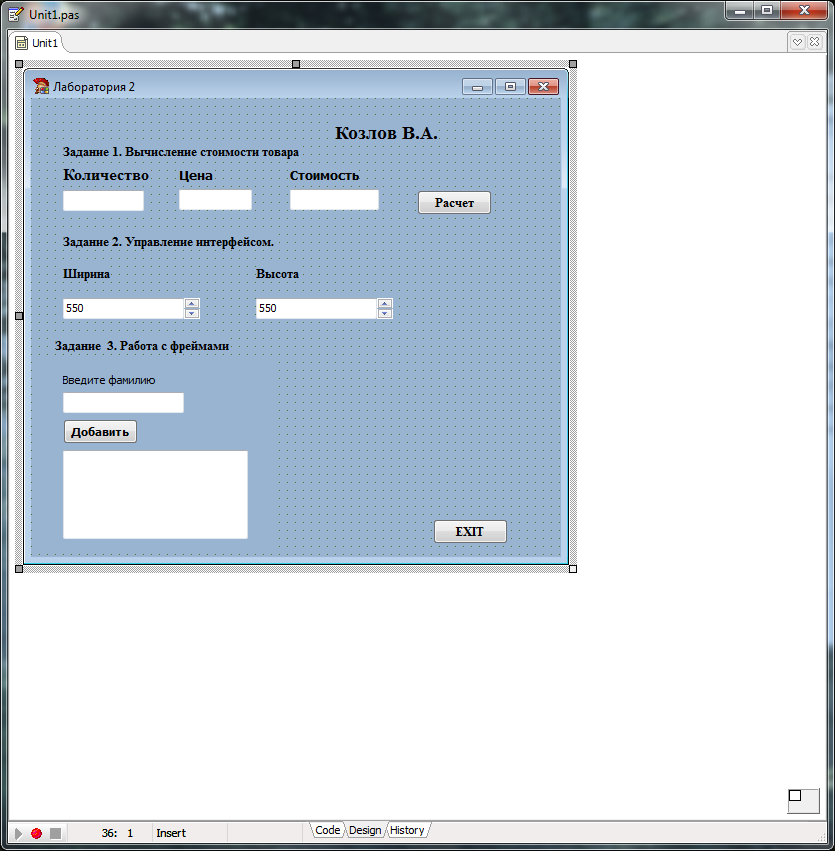
*16.1 Разработать пользовательский интерфейс с использованием компонентов TEdit, TButton и TLabel, с помощью которых осуществляется вычисление арифметических выражений по формулам: Y1 = (X1 + Х2 – Х2)3 – (X3 – Х2)2, где X1, X2, X3 – значения вещественных чисел, введенных в три текстовых поля. Вставить в нижнюю часть формы Вашего (окна) проекта фрейм, разработанный в задании 3. Аналитический вид формулы поместить в интерфейс.*

***Вариант 17.***

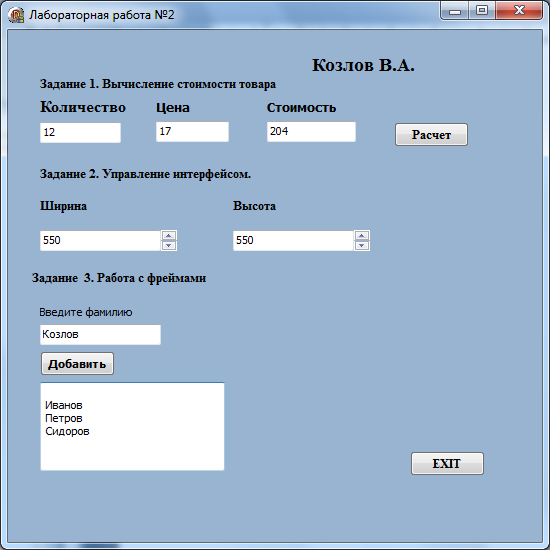
*17.1 Разработать пользовательский интерфейс с использованием компонентов TEdit, TButton и TLabel, с помощью которых осуществляется вычисление арифметических выражений по формулам: Y1 = (X1 – Х2 + Х3)2 – (X3 – Х2/4)2, где X1, X2, X3 – значения вещественных чисел, введенных в три текстовых поля. Вставить в верхнюю часть формы Вашего (окна) проекта фрейм, разработанный в задании 3. Аналитический вид формулы поместить в интерфейс.*

**ПРИЛОЖЕНИЯ**

**Приложение 1.** Интерфейс программы «Лабораторная работа №2»



**Приложение 2.** Интерфейс загрузочного модуля (.exe файла) программы



**Приложение 3.** Программный код

unit Unit1;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls, Buttons, ComCtrls, Unit2;

type

TForm1 = class(TForm)

Label1: TLabel;

Label2: TLabel;

Label4: TLabel;

Edit1: TEdit;

Label5: TLabel;

Edit2: TEdit;

Label6: TLabel;

Edit3: TEdit;

Button1: TButton;

Button2: TButton;

Label7: TLabel;

Label8: TLabel;

Label9: TLabel;

Edit4: TEdit;

Edit5: TEdit;

UpDown1: TUpDown;

UpDown2: TUpDown;

Frame21: TFrame2;

Label3: TLabel;

procedure FormCreate(Sender: TObject);

procedure Button1Click(Sender: TObject);

procedure Button2Click(Sender: TObject);

procedure Edit5Change(Sender: TObject);

procedure Edit4Change(Sender: TObject);

procedure Frame21Button1Click(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Form1: TForm1;

implementation

{$R \*.dfm}

procedure TForm1.Button1Click(Sender: TObject);

var

//Строковые переменные для количества, цены и стоимости

KolS, CenaS, StoimS: string;

//Целочисленные переменные для количества, цены и стоимости

KolInt, CenaInt, StoimInt: integer;

begin

//Считываем из поля Количество введённые данные как строковое значение

KolS := Edit1.Text;

//Считываем из поля Цена введённые данные как строковое значение

CenaS := Edit2.Text;

//Преобразуем символьные значения в целочисленные

KolInt := StrToInt(KolS);

CenaInt := StrToInt(CenaS);

//Вычисляем стоимость по формуле

StoimInt := KolInt \* CenaInt;

//Полученное числовое значение стоимости преобразуем в символьное

StoimS := IntToStr(StoimInt);

//Символьное значение выводим в поле Стоимость

Edit3.Text := StoimS;

end;

procedure TForm1.Button2Click(Sender: TObject);

begin

Close;

end;

procedure TForm1.Edit4Change(Sender: TObject);

begin

form1.Width := StrToInt(Edit4.Text);

end;

procedure TForm1.Edit5Change(Sender: TObject);

begin

form1.Height := StrToInt(Edit5.Text);

end;

procedure TForm1.FormCreate(Sender: TObject);

begin

Form1.Caption:= 'Лабораторная работа №2';

end;

procedure TForm1.Frame21Button1Click(Sender: TObject);

var s: string;

begin

//процедура обработки текстовых данных.

s:=Frame21.Edit1.Text;

Frame21.Memo1.Lines.Add(s);

Frame21.Edit1.Text := '';

end;

end.