УРАЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ
ЛЕСОТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ
УГЛТУ

Т.С. Крайнова

РАЗРАБОТКА ПРОГРАММНЫХ ПРИЛОЖЕНИЙ

Екатеринбург
2014

МИНОБРНАУКИ РОССИИ

ФБГОУ ВПО «УРАЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ЛЕСОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра информационных технологий и моделирования

Т.С. Крайнова

РАЗРАБОТКА ПРОГРАММНЫХ ПРИЛОЖЕНИЙ

Методические указания
по выполнению лабораторно-практических работ
для студентов, обучающихся по направлениям
09.03.03 «Прикладная информатика», 38.03.05 «Бизнес-информатика»
всех форм обучения

Екатеринбург
2014

Печатается по рекомендации методической комиссии ИЭУ.
Протокол № 14 от 09 сентября 2013 г.

Рецензент – Анянова Е.В., канд. с-х. наук, доцент кафедры ИТиМ

Редактор А.Л. Ленская
Оператор компьютерной верстки Т.В. Упорова

Подписано в печать 26.01.15		Поз. 60
Плоская печать	Формат 60×84 1/16	Тираж 10 экз.
Заказ №	Печ. л. 2,56	Цена руб. коп.

Редакционно-издательский отдел УГЛТУ
Отдел оперативной полиграфии УГЛТУ

Введение

Цель методических указаний – научить создавать в среде *Lazarus* законченные программы различного назначения. Большинство рассмотренных примеров демонстрируют назначение компонентов и особенности их применения.

Основные понятия Lazarus

Lazarus – бесплатная среда быстрой разработки с открытым кодом, в которой в качестве языка программирования используется язык *Free Pascal*.

Lazarus представляет собой среду с графическим интерфейсом для быстрой разработки программ и базируется на оригинальной кроссплатформенной библиотеке визуальных компонентов *LCL* (*Lazarus Component Library*). После запуска окно программы имеет вид, показанный на рис. 1.

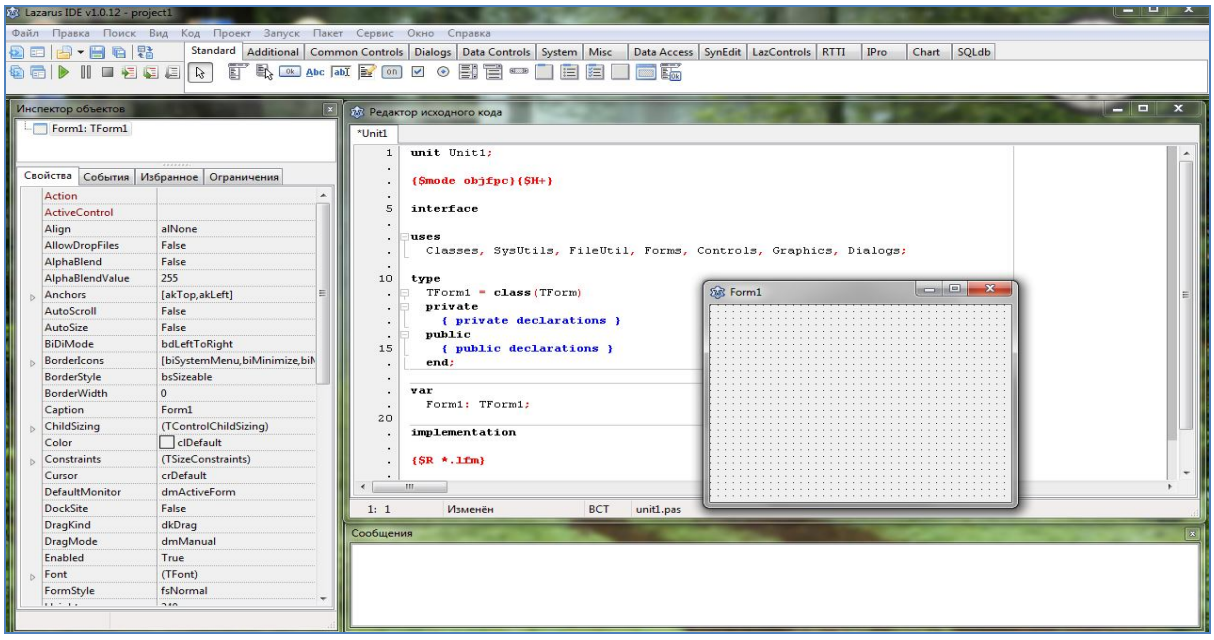


Рис. 1. Окно запуска *Lazarus*

Среда *Lazarus* состоит из нескольких несвязанных окон.

1. <u>Главное окно</u> (рис. 2): позволяет управлять процессом разработки приложения. В нем предусмотрены команды управления файлами, компиляцией, редактированием, окнами и т.д.

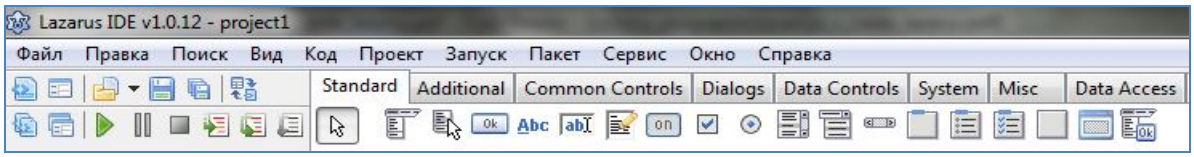


Рис. 2. Главное окно

Главное окно разбито на <u>три функциональных блока</u>.

- *Главное меню* (рис. 3): расположены программы управления файлами, команды управления компиляцией и настройками среды.

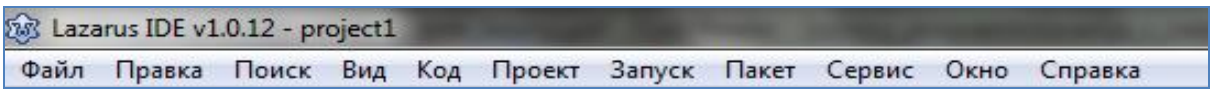


Рис. 3. Главное меню *Lazarus*

- *Панель инструментов* (рис. 4): предоставляет быстрый доступ к основным командам *Главного меню*.

Рис. 4. Панель инструментов

- *Палитра компонентов* (рис. 5): предоставляет доступ к основным компонентам среды разработки, например: поле ввода, надпись, меню, кнопка и т.п.

Рис. 5. Палитра компонентов

2. <u>Окно сообщений</u> (рис. 6): выводятся сообщения компилятора, компоновщика и отладчика.

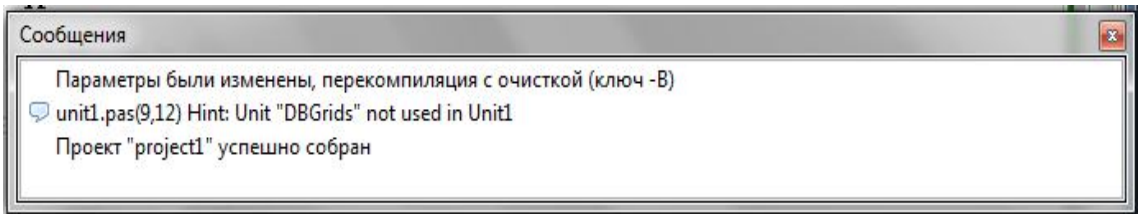


Рис. 6. Окно сообщений

3. <u>Инспектор объектов</u> (рис. 7): позволяет просматривать свойства и методы объекта. В верхней части окна просматривается иерархия объектов, а в нижней части – три вкладки: *Свойства* (перечисляются свойства объекта), *События* (события объекта), *Избранное* (избранные свойства и методы).

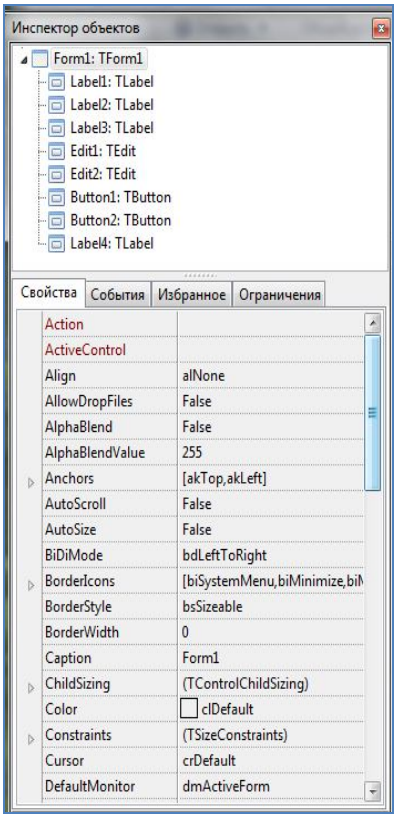


Рис. 7. Инспектор объектов

4. <u>Редактор исходного кода</u> (рис. 8): используется для набора текстов программ.

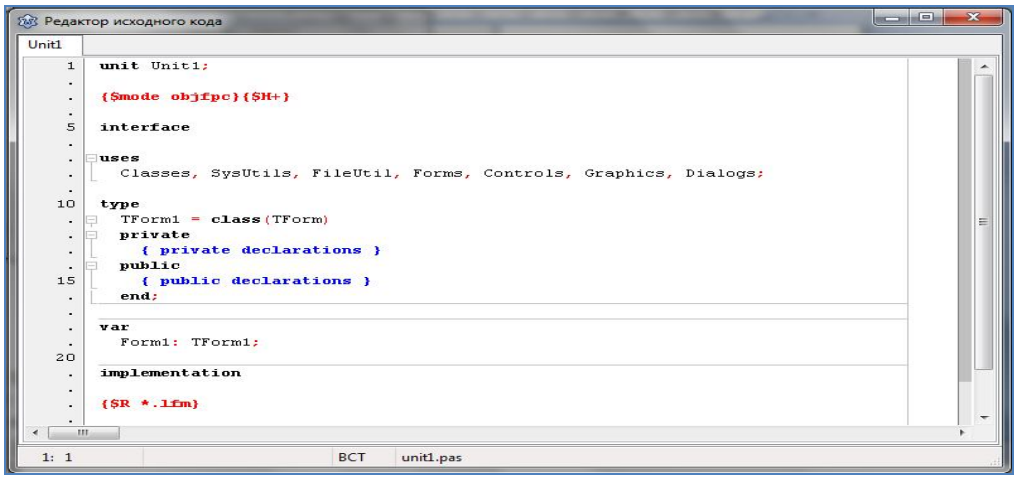


Рис. 8. Окно редактора исходного кода

Проект Lazarus представляет собой набор программных единиц – модулей. Каждая программа включает в себя как минимум один модуль, который содержит описание стартовой формы приложения и поддерживающих его работу процедур. В *Lazarus* каждой форме соответствует свой модуль.

Вид формы подсказывает, как работает приложение. Очевидно, что пользователь вводит в поля редактирования исходные данные и щёлкает по кнопкам. Щелчок на изображении командной кнопки называют *событием*. Таким образом, чтобы программа выполняла некоторую работу в ответ на действие пользователя, программист должен написать процедуру обработки соответствующего события.

В *Lazarus* каждому событию присвоено имя. Например, щелчок кнопкой мыши – это событие *OnClick*, двойной щелчок мышью – это событие *OnDbClick*. В табл.1 приведены описания основных событий.

Таблица 1

События *Lazarus*

Событие	Происходит
OnClick	при щелчке кнопкой мыши
OnDbClick	при двойном щелчке кнопкой мыши
OnMouseDown	при нажатии кнопки мыши
OnMouseUp	при отпускании кнопки мыши
OnMouseMove	при перемещении мыши
OnKeyPress	при нажатии клавиши клавиатуры
OnKeyDown	при нажатии клавиши клавиатуры. События *OnKeyDown* и *OnKeyPress* – это чередующиеся, повторяющиеся события
OnKeyUp	при отпускании нажатой клавиши клавиатуры
OnCreate	при создании объекта (формы, элемента управления), используется при инициализации переменных, выполнении подготовительных действий
OnPaint	при появлении окна на экране в начале работы программы, после появления части окна и в других случаях
OnEnter	при получении элементом управления фокуса
OnExit	при потере элементом управления фокуса

Чтобы приступить к созданию процедуры обработки события, надо сначала в окне *Инспектора объектов* выбрать компонент, для которого создается процедура обработки события. Затем в этом же окне перейти на вкладку *События* (рис. 9).

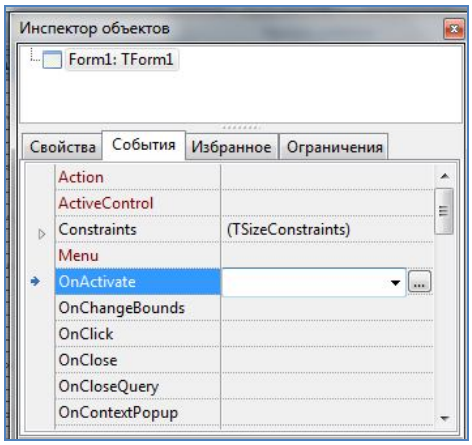


Рис. 9. События, которые может воспринимать компонент
(в данном случае объект *Form1*)

В левой колонке вкладки *События* перечислены имена событий, которые может воспринимать выбранный компонент (объект). Если для события определена (написана) процедура обработки события, то в правой колонке рядом с именем события выводится имя этой процедуры.

Чтобы создать функцию обработки события, необходимо сделать двойной щелчок в поле имени процедуры обработки соответствующего события. В результате этого откроется окно редактора кода, в которое будет добавлен шаблон процедуры обработки события, а в окне *Инспектор объектов* появится имя функции его обработки (рис. 10).

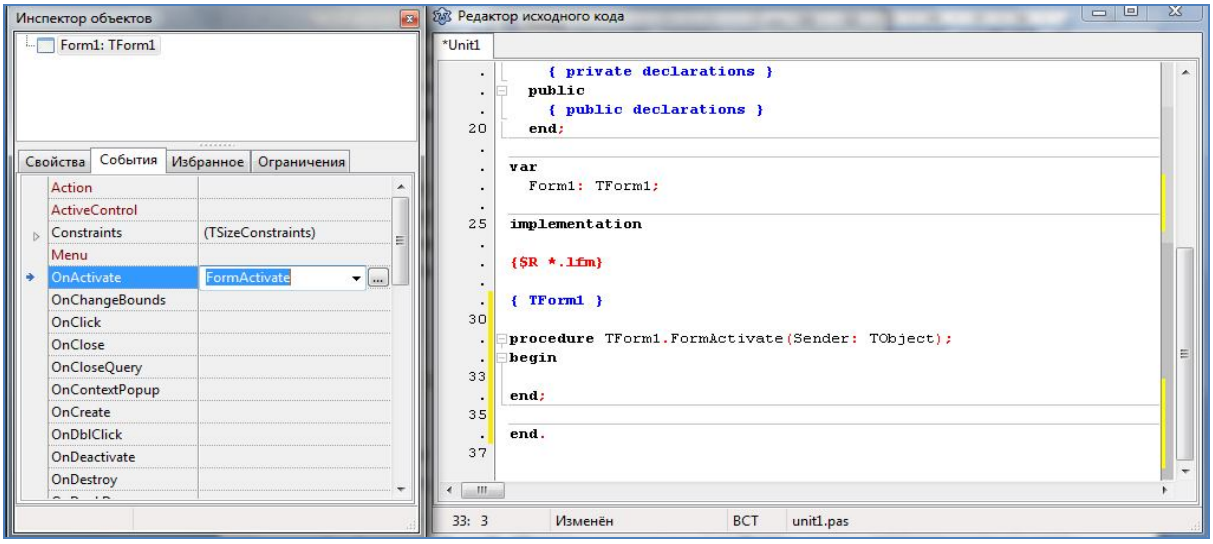


Рис. 10. Шаблон процедуры обработки события *FormActivate*

Lazarus присваивает имя функции обработки события, которое состоит из двух частей. Первая часть имени идентифицирует форму, содержащую объект (компонент), для которого создана процедура обработки события, вторая часть имени идентифицирует сам объект и событие.

Процедуры Lazarus

В языке *FreePascal* основной программной единицей является *процедура*. Структура процедуры в общем виде:

```
Procedure Имя (Список параметров);
        Const
           // объявление констант
        type
           // объявление типов
        var
           // объявление переменных
        begin
           // инструкции программы
        end;
```

Заголовок процедуры состоит из слова ***procedure***, за которым следует имя процедуры, которое используется для вызова процедуры, активации ее выполнения. Если у процедуры есть параметры, то они указываются после имени процедуры, в скобках. Завершается заголовок процедуры символом «точка с запятой».

Если в процедуре используются именованные константы, то они объявляются в разделе объявления констант, который начинается словом ***const***.

В общем виде инструкция объявления именованной константы выглядит следующим образом:

```
константа=значение
```

Здесь

- *константа* – имя константы;
- *значение* – значение константы.

Например:

```
Const Bound = 10;
      pi = 3.1415926;
```

После объявления именованной константы в программе вместо самой константы можно использовать её имя.

За разделом констант следует раздел объявления типов, начинающийся словом ***type***.

После объявления типов идет раздел объявления переменных, в котором объявляются (перечисляются) все переменные, используемые в программе. Раздел объявления переменных начинается словом ***var***.

В общем виде инструкция объявления переменной выглядит так:

Имя: тип

Здесь

- *имя* – имя переменной;
- *тип* – тип данных, для хранения которых предназначена переменная.

Например:

```
Var     A: real;
        C, B: integer;
```

За разделом объявления переменных расположен раздел инструкций. Раздел инструкций начинается словом ***begin*** и заканчивается словом ***end***, за которым следует символ «точка с запятой». В разделе инструкций находятся исполняемые инструкции процедуры, которые часто называют операторами. Одна инструкция от другой отделяется точкой с запятой.

Инструкция присваивания является основной вычислительной инструкцией в записи операторов. В результате выполнения инструкции присваивания значение переменной меняется.

В общем виде инструкция присваивания выглядит так:

Имя := выражение

Здесь

- *Имя* – переменная, значение которой изменяется в результате выполнения инструкции;
- := –символ инструкции присваивания;
- *Выражение* – выражение, значение которого присваивается переменной, имя которой указано слева от символа инструкции присваивания.

Например:

```
Sum := Cena*Kol;
Found := False;
Skidka := 10;
```

Выражение состоит из операндов и операторов. Операторы находятся между операндами и обозначают действия, которые выполняются над операндами. В качестве операндов можно использовать переменную, константу, функцию или другое выражение. Основные алгебраические операторы приведены в табл. 2.

Таблица 2

Алгебраические операторы

Оператор	Действие
+	сложение
-	вычитание
*	умножение
/	деление
DIV	деление нацело
MOD	вычисление остатка от деления

Тип выражения определяется типом операндов, входящих в выражение, и зависит от операций, выполняемых над ними. В табл. 3 приведены правила определения типа выражения в зависимости от типа операндов и вида оператора.

Таблица 3

Правила определения типа выражения

Оператор	Тип операндов	Тип выражения
*, +, -	хотя бы один из операндов *real*	*real*
*, +, -	оба операнда *integer*	*integer*
/	*real* или *integer*	**всегда** *real*
DIV, MOD	всегда integer	**всегда** *integer*

Функции преобразования (табл. 4) наиболее часто используются в инструкциях, обеспечивающих ввод и вывод информации.

Таблица 4

Функции преобразования

Функция	Значение функции
Chr(n)	символ, код которого равен n
IntToStr(k)	представление целого числа в символьном типе
FloatToStr(n)	представление вещественного числа в символьном типе
StrToInt(s)	представлениесимвольного значения в числовом целом типе
StrToFloat(s)	представлениесимвольного значения в вещественном типе
Round(n)	целое, полученное путем округления n по известным правилам
Trunk(n)	целое, полученное путем отбрасывания дробной части n
Frac(n)	дробное, представляющее собой дробную часть вещественного n
Int(n)	дробное, представляющее собой целую часть вещественного n

Символьная информация может быть представлена как отдельными символами, так и строками (последовательностью символов). Для хранения и обработки символов поддерживается тип *Char*. Перед служебным символом ставится оператор #. В табл. 5 приведены коды некоторых служебных символов.

Таблица 5

Служебные символы

Код символа	Символ
9	табуляция
13	новая строка
8	клавиша<Back Space>
32	пробел

Например, часто используемый при записи сообщений символ «новая строка» записывается так: #13. Такой способ записи, как правило, используют для записи символов, которые во время набора программы нельзя ввести с клавиатуры.

Создание проекта

Из главного меню **Файл** выбрать команду **Создать…**. В появившемся диалоговом окне (рис. 11) выбрать из списка слово **Приложение** и нажать кнопку **Ok**. Результатом этих действий будет появление *окна формы* и *окна редактора программного кода*.

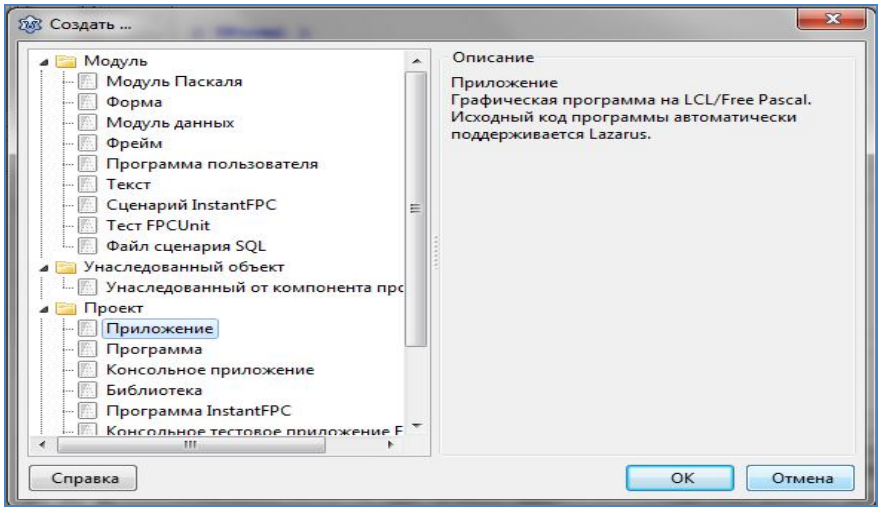


Рис. 11. Окно создания приложения

Сохранение проекта

Проект – это набор файлов, используя которые компилятор создает исполняемый файл программы (*exe-файл*). В простейшем случае проект состоит из файла главного модуля (*lpr-файл*), файла ресурсов (*res-файл*), файла описания формы (*lfm-файл*), файла модуля формы (*pas-файл*), файла конфигурации (*cfg-файл*).

Чтобы сохранить проект, нужно из меню **Файл** выбрать команду **Сохранить как..**. Если проект еще ни разу не был сохранен, то *Lazarus* сначала предложит сохранить проект, поэтому на экране появится окно *Сохранить проект*. В этом окне надо выбрать папку, предназначенную для файлов проекта. После нажатия кнопки **Сохранить**, появится следующее окно, в котором сохраняется *Модуль проекта*. Имена файлов модуля (*pas-файл*) и проекта (*lpr-файл*) должны быть разными.

Лабораторная работа № 1

Цель: знакомство с компонентами *Edit*, *Label*, *Button*, использование сообщений для вывода результата.

Работа над новым проектом, так называется разрабатываемое приложение, начинается с создания стартовой формы (появляетсяпри запуске *Lazarus*) и добавления к форме необходимых компонентов (полей ввода и вывода текста, командных кнопок и других элементов управления). Для того чтобы добавить на форму компонент, необходимо в *Палитре компонентов* выбрать соответствующий компонент, щелкнув левой кнопкой мыши на его пиктограмме, далее, установив курсор в ту точку формы, в которой должен быть левый верхний угол компонента, еще раз щелкнуть кнопкой мыши. В результате на форме появится компонент стандартного размера. Каждому компоненту *Lazarus* присваивается имя, которое состоит из названия компонента и его порядкового номера. Например, Edit1 или ComboBox3.

Для просмотра и изменения значений свойств формы и её компонентов используется окно *Инспектор объектов*. В верхней части окна *Инспектор объектов* указано имя объекта, значения свойств которого отображаются в данный момент. В левой колонке вкладки Свойства перечислены свойства объекта, а в правой – указаны их значения.

Form–является основой программы. Свойства формы (табл. 6) определяют вид окна программы.

Таблица 6

Свойства объекта *Form*

Свойство	Описание
Name	имя формы, используется для управления формой и доступа к компонентам формы
Caption	текст заголовка
Width	ширина формы
Height	высота формы
Top	расстояние от верхней границы формы до верхней границы экрана
Left	расстояние от левой границы формы до верхней границы экрана
BorderStyle	вид границы: обычная (*bsSizeable*), тонкая (*bsSingle*) или отсутствовать (*bsNone*)

Окончание табл. 6

Свойство	Описание
BorderIcons	кнопки управления окном. Свойство *biSystemMenu* определяет доступность *Системного меню*, *biMinimaze* – кнопки *Свернуть* и *biMaximaze* - *Развернуть*
Icon	Значок в заголовке диалогового окна, обозначающего кнопку вывода системного меню
Color	цвет фона
Font	шрифт

Edit (тип - String)–представляет собой поле ввода - редактирования строки символов, располагается на вкладке *Standart*. Вид пиктограммы представлен на рис. 12. В табл. 7 перечислены свойства компонента.

Рис. 12. Компонент Edit

Таблица 7

Свойства компонента *Edit*

Свойство	Описание
Name	имя компонента
Text	текст, находящийся в поле ввода и редактирования (обычно удаляется)
Left	расстояние от левой границы компонента до левой границы формы
Top	расстояние от верхней границы компонента до верхней границы формы
Height	высота поля
Width	ширина поля
Font	шрифт, используемый для отображения вводимого текста
ParentFont	признак наследования компонентом характеристик шрифта формы, на которой находится компонент; если значение свойства *True*, то при изменении свойства *Font* формы автоматически меняется значение свойства *Font* компонента

Label(тип - String)–предназначен для вывода текста на поверхность формы, располагается на вкладке *Standart*. Вид пиктограммы представлен на рис. 13. Свойства компонента *Label* перечислены в табл. 8.

Рис. 13. Компонент Label

Таблица 8

Свойства компонента *Label*

Свойство	Описание
Name	имя компонента для доступа к его свойствам
Caption	отображаемый текст
Font	шрифт, используемый для отображения текста
ParentFont	признак наследования компонентом характеристик шрифта формы; если *True*, текст выводится шрифтом, установленным для формы
Alignment	выравнивание текста
AutoSize	признак того, что размер поля определяется его содержимым
Left	расстояние от левой границы поля до левой границы формы
Top	расстояние от верхней границы поля до верхней границы формы
Height	высота поля
Width	ширина поля
WordRap	признак того, что слова, которые не помещаются в текущей строке, автоматически переносятся на следующую строку

Button– представляет собой командную кнопку, располагается на вкладке *Standart*. Вид пиктограммы представлен на рис. 14. Свойства компонента перечислены в табл. 9.

Рис. 14. Компонент Button

Таблица 9

Свойства компонента *Button*

Свойство	Описание
Name	имя компонента
Caption	текст на кнопке
Enabled	признак доступности кнопки: *True* – доступна, *False* – недоступна
Left	расстояние от левой границы кнопки до левой границы формы
Top	расстояние от верхней границы кнопки до верхней границы формы
Height	высота кнопки
Width	ширина кнопки

Окно ввода

Окно ввода – это стандартное диалоговое окно, которое появляется на экране в результате вызова функции **InputBox**. Значение функции *InputBox* – строка, которую ввел пользователь.

В общем виде инструкция ввода данных с использованием функции *InputBox* выглядит так:

Переменная:=InputBox(Заголовок, Подсказка, Значение)

Здесь

- *Переменная* – переменная строкового типа, значение которой должно быть получено от пользователя;
- *Заголовок* – текст заголовка окна ввода;
- *Подсказка* – текст поясняющего сообщения;
- *Значение* – текст, который будет находиться в поле ввода, когда окно появится на экране.

Например, InputBox('Фунты-килограммы', 'Введите вес в фунтах','0'). Окно ввода, соответствующее этой инструкции, приведено на рис. 15.

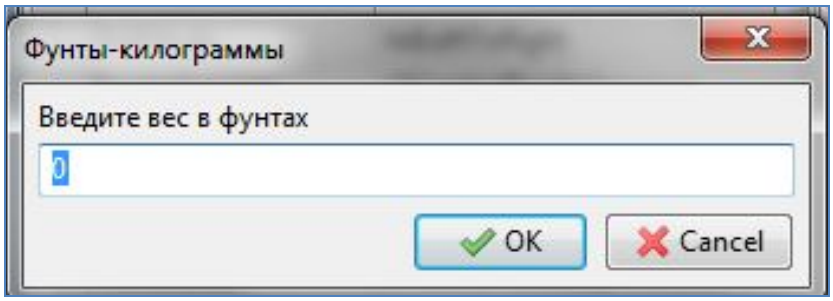


Рис. 15. Пример окна ввода

Если во время работы пользователь введет строку и щёлкнет по кнопке **Ok**, то значением функции *InputBox* будет введенная строка. Если будет сделан щелчок по кнопке **Cancel**, то значением функции будет строка, переданная функции в качестве параметра *Значение*.

Вывод результатов в окно сообщений

Окна сообщений используются для привлечения внимания пользователей. С помощью окна сообщения программа может, к примеру, проинформировать об ошибке в исходных данных или запросить подтверждение выполнения необратимой операции, например, удаление файла.

Вывести на экран окно с сообщением можно с помощью процедуры *ShowMessage* или функции *MessageDlg*.

Процедура **ShowMessage** выводит на экран окно с текстом и командной кнопкой **Ok**. В общем виде инструкция вызова процедуры *ShowMessage* выглядит так:

ShowMessage (Сообщение)

Здесь *Сообщение* – текст, который будет выведен в окне. Например, на рис. 16 дан вид окна сообщения, полученного в результате выполнения инструкции: ShowMessage ('Введите вес в фунтах').

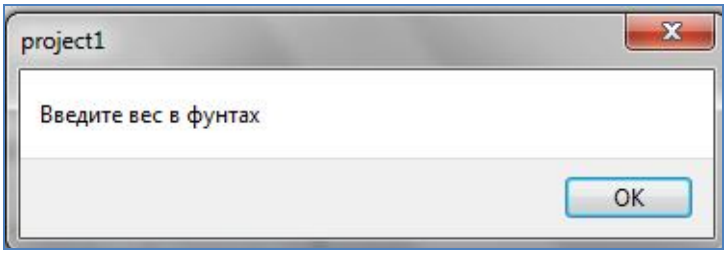


Рис. 16. Пример окна сообщений

Функция **MessageDlg** позволяет поместить в окно с сообщением один из стандартных значков, например “Внимание”, задать количество и тип командных кнопок и определить, какую из кнопок нажал пользователь в результате выполнения инструкции.

Значение функции *MessageDlg* – число, проверив значение которого, можно определить, выбором какой командной кнопки был завершен диалог.

В общем виде обращение к функции *MessageDlg* выглядит так:

Выбор:=MessageDlg(Сообщение, Тип, Кнопки, КонтекстСправки)

Здесь

- *Сообщение* – текст сообщения;
- *Тип* – тип сообщения (информационное или предупреждающее о критической ошибке). Каждому типу сообщения соответствует определенный значок. Тип сообщения задается именованной константой (табл. 10);
- *Кнопки* – список кнопок, отображаемых в окне сообщения. Список может состоять из несколько разделенных запятыми именованных констант (табл. 11). Весь список заключается в квадратные кнопки;
- *КонтекстСправки* – параметр, определяющий раздел справочной системы, который появится на экране, если пользователь нажмет клавишу <F1>. Если вывод справки не предусмотрен, то значение параметра *КонтекстСправки* должно быть равно нулю.

Таблица 10

Константы функции *MessageDlg*

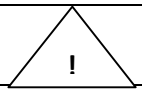
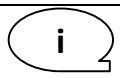
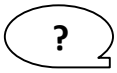
Константа	Тип сообщения	Значок
mtWarning	внимание	
mtError	ошибка	
mtInformation	информация	
mtConfirmation	подтверждение	
mtCustom	обычное	без значка

Таблица 11

Константы функции *MessageDlg*

Константа	Кнопка	Константа	Кнопка	Константа	Кнопка
mbYes	Yes	mbAbort	Abort	mbCancel	Cancel
mbNo	No	mbRetry	Retry	mbHelp	Help
mbOk	OK	mbIgnore	Ignore	mbAll	All

Например, для того чтобы в окне сообщения появились кнопки **Ok** и **Cancel**, список *Кнопки* должен быть таким:

[mbOk, mbCancel]

Наиболее часто используются в диалоговых окнах комбинации командных кнопок: mbOkCancel, mbYesNoCancel, mbAbortRetryIgnore.

Например, на рис. 17 приведено окно, выведенное в результате выполнения инструкции:

MessageDlg('Файл будет удален.', mtWarning, [mbOk, mbCancel], 0)

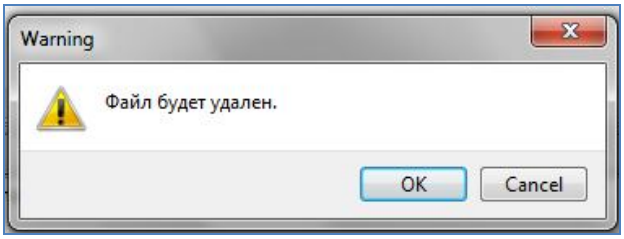


Рис. 17. Пример окна сообщения

Задание к Лабораторной работе № 1

Задача. Разработать приложение, которое вычисляет скорость, с какой спортсмен пробежал дистанцию.

1. Создать папку Lab1.

2. Запустить *Lazarus* и создать форму, имеющую вид, приведенный на рис. 18.

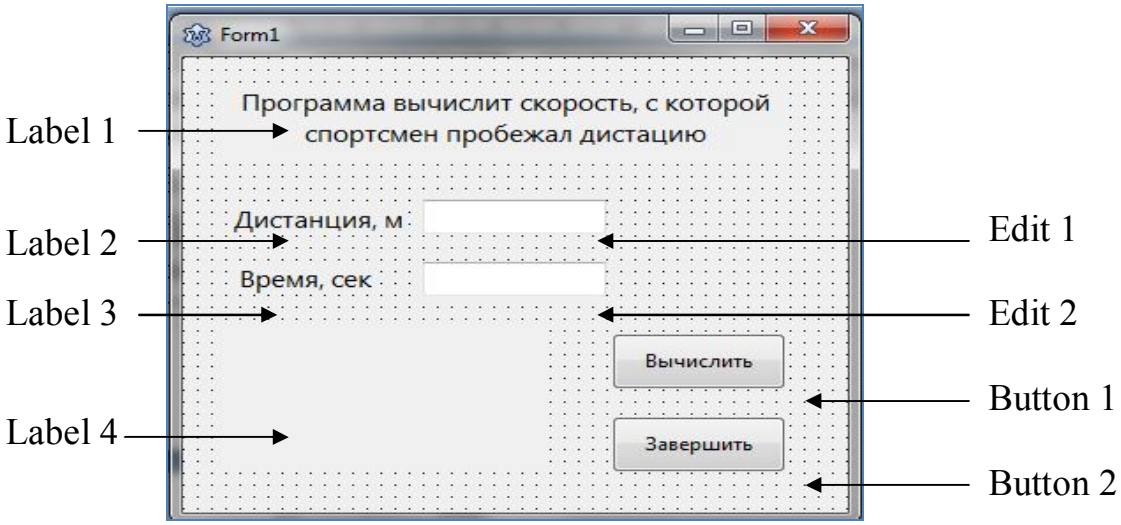


Рис. 18. Форма программы *Скорость бега*

3. Сохранить проект.
4. Указать свойства объекта *Form*:
   - граница формы – тонкая;
   - кнопки *Свернуть*, *Развернуть* недоступны;
   - размер шрифта – 12 пт.
5. Написать процедуру *OnClick* для кнопки *Вычислить*:

```
procedure TForm1.Button1Click(Sender: TObject);
var
dist :integer; // дистанция, метров
   t:  real;    // время
   v: real;       // скорость
begin

// получить исходные данные из полей ввода
dist := StrToInt(Edit1.Text);
t := StrToFloat(Edit2.Text);

// вычисление
v := dist/t;
// вывод результата
Label4.Caption := 'Дистанция: '+ Edit1.Text + ' м' + #13 + 'Время: ' +
FloatToStr(t) + ' сек ' + #13 + 'Скорость: ' + FloatToStrF(v,ffFixed,4,2) + ' м/с';
end;
```

6. Написать процедуру для кнопки *Завершить*:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
Form1.Close;
end;
```

7. Проверить работу приложения.

После нескольких запусков программы *Скорость бега* возникает желание внести изменения в программу. Например, чтобы после ввода дистанции и нажатия клавиши <Enter> курсор переходил в поле *Время*. Для этого необходимо добавить процедуры обработки событий *OnKeyPress* для компонентов *Edit1* и *Edit2*.

```
// нажатие клавиши в поле Дистанция
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
   // Key - символ, соответствующий нажатой клавише.
   // Если символ недопустимый, то процедура заменяет его
   // на символ с кодом 0. В результате этого символ в поле
   // редактирования не появляется и у пользователя создается
   // впечатление, что программа не реагирует на нажатие некоторых
// клавиш.
case Key of
    '0'..'9':  ;// цифра
    #8 :   ; // <Backspace>
    #13 : Edit2.SetFocus   ; // <Enter>
```

```
    // остальные символы - запрещены
elseKey :=Chr(0);  // символ не отображать
end;
end;

// нажатие клавиши в поле Время
procedure TForm1.Edit2KeyPress(Sender: TObject; var Key: Char);
begin
case Key of
'0'..'9':   ;// цифра
',','.' :// точка или запятая
begin
if Key = '.' then Key := ','; // подменим точку запятой

// проверим, есть ли в поле Edit2 запятая
if Pos(',',Edit2.Text) <> 0 // запятая уже есть
thenKey:=Char(0);      // вторую запятую не отображать
end;
    #8 :; // <Backspace>
// остальные символы - запрещены
elseKey :=Chr(0);  // символ не отображать
end;
end;
```

После внесения изменений сохранить проект и проверить работу.

Дополнительные задания

1. В проекте *Скорость бега* обеспечить переход по клавише <Enter> с компонента *Edit2* на кнопку *Вычислить*.

2. Разработать проект, обеспечивающий проверку вводимого данного на отрицательность значения. Оповестить пользователя об ошибке в диалоговом окне *MessageDlg*.

3. Разработать проект, обеспечивающий *Проверку числа* на четность, положительность, кратность трём. При вводе числа использовать *InputBox*, а вывод результата – через *ShowMessage*.

4. Написать программу, которая вычисляет *Стоимость телефонного разговора*. Пользователь вводит длительность разговора и номер дня недели. Если день недели – суббота или воскресенье, стоимость уменьшается на скидку. Цена минуты и величина скидки (25 %) задаются в программе как константы.

Лабораторная работа № 2

Цель: знакомство с компонентами *ListBox*,*CheckBox*, *ComboBox*.

ListBox (рис. 19) представляет собой список, в котором можно выбрать нужный элемент. Пиктограмма компонента располагается на вкладке *Standart*. Свойства компонента приведены в табл. 12.

Список может быть сформирован во время создания формы или во время работы программы.

Для формирования списка во время создания формы надо в окне *Инспектора объектов* выбрать свойство *Items*и щёлкнуть по кнопке запуска редактора списка строк. В открывшемся диалоговом окне *Диалог ввода строк* ввести список, набирая каждый элемент списка в отдельной строке. После ввода очередного элемента списка для перехода к новой строке необходимо нажать клавишу <Enter>. После ввода последнего элемента клавишу <Enter> нажимать не надо. Завершив ввод списка, следует щёлкнуть по кнопке *Ok*.

Рис. 19. Компонент ListBox

Таблица 12

Свойства компонента *ListBox*

Свойство	Определяет
Name	имя компонента, используется для доступа к свойствам компонента
Items	элементы списка
Count	количество элементов списка
ItemIndex	номер выбранного элемента списка. Номер первого элемента списка равен нулю. Если ни один из элементов не выбран, то значение свойства равно минус единице
Left	расстояние от левой границы списка до левой границы формы
Top	расстояние от верхней границы списка до верхней границы формы
Height	высота поля списка
Width	ширина поля списка
Font	шрифт, используемый для отображения элементов списка
ParentFont	признак наследования свойств шрифта родительской формы

CheckBox (рис. 20) представляет собой независимую кнопку (переключатель). Свойства компонента приведены в табл. 13.

Рис. 20. Компонент CheckBox

Таблица 13

Свойства компонента *CheckBox*

Свойство	Определяет
Name	имя компонента, используется для доступа к свойствам компонента
Caption	текст, который находится справа от флажка
Checked	состояние, внешний вид флажка: *True* – установлен, *False* – флажок сброшен
Left	расстояние от левой границы флажка до левой границы формы
Top	расстояние от верхней границы флажка до верхней границы формы
Height	высота поля поясняющего текста
Width	ширина поля поясняющего текста
Font	шрифт, используемый для отображения поясняющего текста
ParentFont	признак наследования свойств шрифта родительской формы

ComboBox (рис. 21) представляет собой поле со списком, в котором можно выбрать нужный элемент. Варианты заполнения списка значениями как у компонента *ListBox*. Свойства компонента приведены в табл. 14.

Рис. 21. ComboBox

Таблица 14

Свойства компонента *ComboBox*

Свойство	Определяет
Name	имя компонента, используется для доступа к свойствам компонента
Text	текст, находящийся в поле ввода - редактирования
Items	элементы списка
Count	количество элементов списка
ItemIndex	номер выбранного элемента списка. Номер первого элемента списка равен нулю. Если ни один из элементов не выбран, то значение свойства равно минус единице
Left	расстояние от левой границы списка до левой границы формы
Top	расстояние от верхней границы списка до верхней границы формы
Height	высота поля списка
Width	ширина поля списка
Font	шрифт, используемый для отображения элементов списка
ParentFont	признак наследования свойств шрифта родительской формы

При работе со списками используется инструкция *Case*, позволяющая эффективно реализовывать множественный выбор. В общем виде записывается следующим образом:

```
Case Селектор of
        список1:
                {инструкции 1}
        …….
        списокN:
                {инструкции N}
end;
```

Здесь

- *Селектор* – выражение, значение которого определяет дальнейшее выполнение программы;
- *СписокN* – список констант. Если константы представляют собой диапазон чисел, то вместо списка можно указать первую и последнюю константу диапазона, разделив их двумя точками. Например, список 1, 2, 3, 4, 5, 6 может быть заменен диапазоном 1..6.

Задание к Лабораторной работе № 2

Задача. Написать программу *Конвертер*, которая переводит массу товара из фунтов в килограммы в соответствии со значением коэффициента выбранной страны.

1. Создать папку Lab2.
2. Поместить на форму компоненты (рис. 22) и сохранить проект.

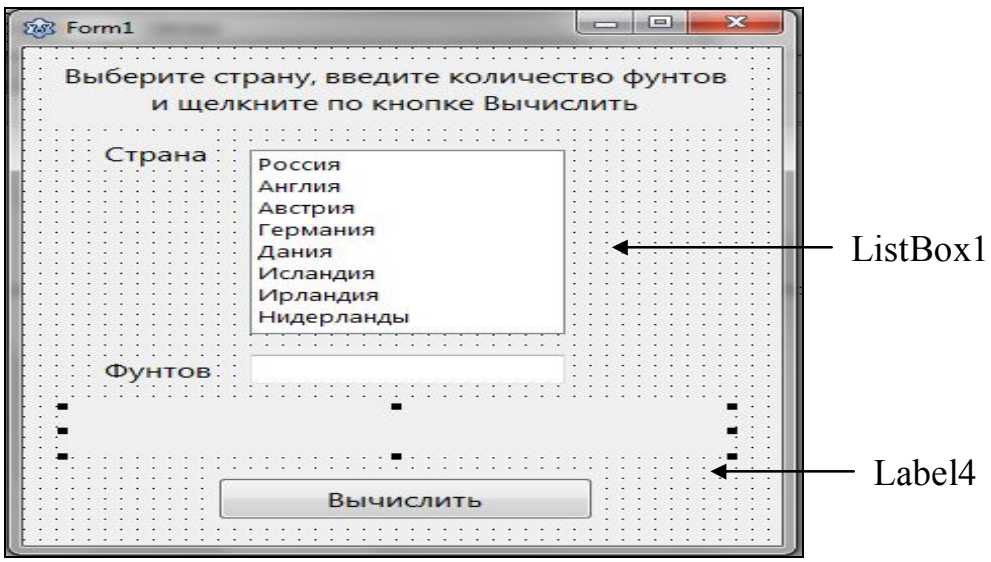


Рис. 22. Интерфейс формы *Конвертер*

3. Для кнопки *Вычислить* написать обработку события *OnClick*.

```
procedure TForm1.Button1Click (Sender: TObject);
var
funt:real; { вес в фунтах }
```

```
kg:real;   { вес в килограммах }
k:real;    { коэффициент пересчета }
begin
case ListBox1.ItemIndex of
0: k:=0.4059;  { Россия }
     1: k:=0.453592;{ Англия }
     2: k:=0.56001; { Австрия }
3..5,7:k:=0.5; { Германия, Дания, Исландия, Нидерланды }
6: k:=0.31762; { Италия }
end;
funt:=StrToFloat(Edit1.Text);
kg:=k*funt;
Label4.caption:=Edit1.Text + ' фунт. - '+ FloatToStrF(kg,ffFixed,6,3)+ ' кг.';
end;
```

4. Проверить работу приложения.

Дополнительные задания

1. Написать программу, добавляющую в компонент *ListBox* записи с помощью диалогового окна *InputBox*. Количество записей в *ListBox* определяется программно.

2. Написать программу, рассчитывающую общую сумму заказа на 1 человека в кафе: добавить на форму четыре компонента *ListBox* (холодные закуски, горячие блюда, напитки, десерты), заполнить произвольными значениями. Вычисленную стоимость выводить в *Label*.

3. Написать программу, рассчитывающую стоимость проезда в одну сторону или с возвратом: добавить на форму компоненты *ComboBox* со списком названий остановок и *CheckBox*, обозначающий проезд «туда и обратно». Ответ выводить в компоненте *Label*.

4. Написать программу, которая вычисляет доход по вкладу. Программа должна обеспечить расчет простых процентов (*CheckBox1*) – в конце срока вклада и сложных (*CheckBox2*) – ежемесячно, которые прибавляются к текущей (накопительной сумме вклада, в следующем месяце проценты начисляются на новую сумму. При расчете использовать значения Суммы вклада (руб.), Срока (мес.), Процентной ставки.

Лабораторная работа № 3

Цель: работа с массивами, знакомство с компонентами *Memo*, *StringGrid*.

Массив – это структура данных, представляющая собой набор переменных одинакового типа, имеющих общее имя. Массивы удобно использовать для хранения однородной по своей природе информации.

Массив, как и любая переменная программы, должен быть объявлен в разделе объявления переменных. В общем виде объявление массива выглядит так:

*Имя_массива:***array***[нижний_индекс..верхний_индекс]***of***тип*

Здесь

- *Имя* – имя массива;
- *array* – зарезервированное слово, обозначающее, что объявляемое имя является именем массива;
- *нижний_индекс* и *верхний_индекс* – целые константы, определяющие диапазон изменения индекса элементов массива и, неявно, количество элементов (размер) массива;
- *тип* – тип элементов массива.

Примеры объявления массивов:

```
temper: array [1..31] of real;
koef: array [0..10] of integer;
name: array [1..30] of string [25];
```

Ввод массива:

Под вводом массива понимается процесс получения от пользователя во время работы программы значений элементов массива. Для ввода массива удобно использовать компонент *StringGrid*. Для перевода текстовых данных в числовые используется функция *StrToInt (StrToFloat)*.

Например, задать одинарный массив из пяти элементов:

```
for i:= 1 to 5 do
  if Length(StringGrid1.Cells[i-1,0]) <> 0
   then a[i] := StrToInt(StringGrid1.Cells[i-1,0])else a[i] := 0;
```

Вывод массива

Если в программе необходимо вывести список всех элементов массива, то для этого удобно использовать инструкцию *for*. Например, вывести одинарный массив из пяти элементов в *Label*:

```
for i:=1 to 5 do
Label1.Caption:=  Label1.Caption+IntToStr(a[i]);
```

Memo (рис. 23) представляет собой компонент редактирования текста, который может состоять из нескольких строк, поэтому используется для символьного массива. Свойства компонента приведены в табл. 15.

Рис. 23. Пиктограмма Memo

Таблица 15

Свойства компонента *Memo*

Свойство	Определяет
Name	имя компонента, используется для доступа к свойствам компонента
Text	текст, находящийся в поле *Memo*
Lines	текст, находящийся в поле *Memo*. Рассматривается как совокупность строк. Доступ к строке осуществляется по номеру
Lines.Count	количество строк текста в поле *Memo*
Left	расстояние от левой границы поля до левой границы формы
Top	расстояние от верхней границы поля до верхней границы формы
Height	высота поля
Width	ширина поля
Font	шрифт, используемый для отображения вводимого текста
ParentFont	признак наследования свойств шрифта родительской формы

При использовании компонента *Memo* для ввода массива значение каждого элемента массива следует вводить в отдельной строке и после ввода каждого элемента массива нажимать клавишу <Enter>.

Получить доступ к находящейся в поле *Memo* строке текста можно с помощью свойства *Lines*, указав в квадратных скобках номер нужной строки (строки нумеруются с нуля).

Основной цикл процедуры ввода символьного массива из компонента *Memo* может выглядеть так:

```
For i := 1 to size do
a[i] := Memo1.Lines[i];
```

Здесь

- *size* – именованная константа, определяющая размер массива;
- *a* – массив;
- *Memo1* – имя *Memo*-компонента;
- *Lines* – свойство компонента *Memo*, представляющее собой массив, каждый элемент которого содержит одну строку находящегося в поле *Memo* текста.

StringGrid (рис. 24) представляет собой таблицу, ячейки которой содержат строки символов. Компонент находится на вкладке *Additional*. В табл.16 перечислены свойства компонента *StringGrid*.

Рис. 24. Пиктограмма StringGrid

Таблица 16

Свойства компонента *StringGrid*

Свойство	Определяет
Name	имя компонента, используется для доступа к свойствам компонента
ColCount	количество колонок таблицы
RowCount	количество строк таблицы
Cells	ячейка таблицы, находящаяся на пересечении столбца *col* и строки *row*, определяется элементом *cells[col, row]*
FexedColls	количество зафиксированных слева колонок таблицы
FixedRows	количество зафиксированных сверху колонок таблицы
Options.goEditing	признак допустимости редактирования содержимого ячеек таблицы, принимает значения *False*, *True*
Options.goTab	разрешает (*True*) и запрещает (*False*) использование клавиши <Tab> для перемещения курсора в следующую ячейку
Otions.goAlwaysShowEditor	признак нахождения компонента в режиме редактирования. Если значение свойства *False*, то для того, чтобы в ячейке появился курсор, надо начать набирать текст, нажать клавишу F2 или сделать щелчок мышью
DefaultColWidth	ширина колонок таблицы
DefaultRowHeight	высота колонок таблицы
GridLineWidth	ширина линий, ограничивающих ячейки таблицы
Left	расстояние от левой границы поля таблицы до левой границы формы
Height	высота поля таблицы
Width	ширина поля таблицы
Font	шрифт, используемый для отображения содержимого ячеек таблицы
ParentFont	признак наследования характеристик шрифта формы

Задание к Лабораторной работе № 3

Задача 1. При щелчке по кнопке *Button1* выполняется вывод значений элементов массива из поля *Memo1* в компоненте *ShowMessage*, число введенных в *Memo1* строкне должно превышать 5 (пяти).

1. Создать папку Lab 31.
2. Поместить на форму компоненты (рис. 25).

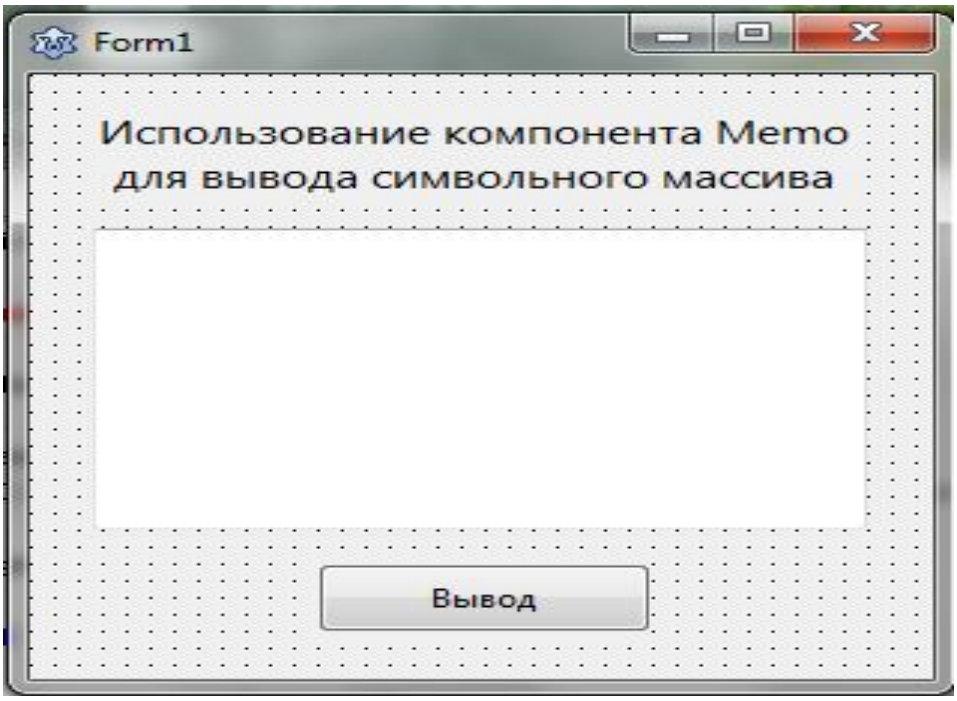


Рис. 25. Интерфейс программы

3. Ввести текст модуля события *OnClick* для кнопки *Button1*:

```
procedure TForm1.Button1Click(Sender: TObject);
const
   SIZE=5;  //размермассива
var
a:array[1..SIZE]of string[30]; // массив
n: integer; // количество строк, введенных в поле Memo
   i:integer;  // индекс элемента массива
st:string;
begin
n:=Memo1.Lines.Count;
// в поле Memo текста нет
ifn = 0 then
begin
ShowMessage('Исходные данные не введены!');
Exit;   // выход из процедуры обработки события
end;

// вполеMemoестьтекст
ifn>SIZEthen
begin
ShowMessage('Количество введенных строк превышает размер массива');
       n:=SIZE; // будем вводить только первые SIZE строк
end;
```

```
for i:=1 to n do
a[i]:=Form1.Memo1.Lines[i-1]; // строки Memo пронумерованыснуля
// вывод массива в окно сообщения
ifn> 0 then
begin
st:='Введенный массив:'+#13;
for i:=1 to n do
st:=st+IntToStr(i)+'  '+ a[i]+#13;
ShowMessage(st);
end;
end;
```

4. Проверить работу приложения.

Задача 2. Вычислить среднее арифметическое значение элементов одинарного массива A(5). Диалоговое окно программы приведено на рис. 26.

1. Создать папку Lab32.

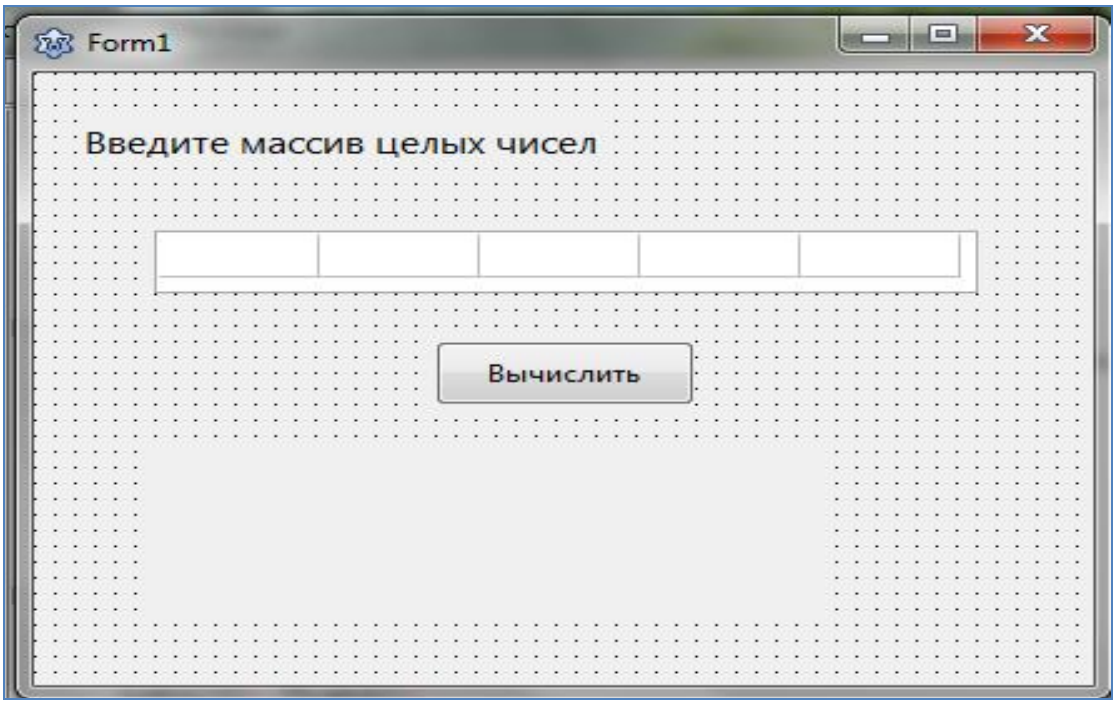


Рис. 26. Ввод и обработка массива

2. Поместить на форму компоненты и указать значения свойств *StringGrid1* (табл. 17).

Таблица 17

Значения свойств для *StringGrid1*

Свойство	Значение
ColCount	5
FixedCols	0
FixedRows	0
RowCount	1
DefaultRowHeight	24
Height	24
DefaultColWidth	64
Width	328
Options.goEditing	True
Options.AlwaysShowEditing	True
Options.goTabs	True

3. Ввести текст модуля для события *OnClick* компонента *Button1*:

```
procedure TForm1.Button1Click(Sender: TObject);
var
a : array[1..5] of integer; // массив
summ: integer;              // сумма элементов
sr:  real;                  // среднее арифметическое
i: integer;                 // индекс
begin
// ввод массива
// считаем, что если ячейка пустая, то соответствующий
// ей элемент массива равен нулю
fori:= 1 to 5 do
if Length(StringGrid1.Cells[i-1,0]) <> 0
then a[i] := StrToInt(StringGrid1.Cells[i-1,0])
else a[i] := 0;

  // обработка массива
summ := 0;
for i :=1 to 5 do
summ := summ + a[i];
sr := summ / 5;

  // вывод результата
Label2.Caption  :='Суммаэлементов: ' + IntToStr(summ) + #13+ 'Среднее
арифметическое: ' + FloatToStr(sr);
end;
```

4. Проверить работу программы.

Дополнительные задания

1. Найти максимальный элемент массива B(7) и вывести номер этого элемента в *Label*.

2. Вывести в *ShowMessage* все отрицательные четные элементы массива F(3,6).

3. Найти количество одинаковых элементов в массиве K(5, 3), равных образцу, заданному в компоненте *Edit*. Ответ вывести в компоненте *Label*.

4. Вывести в компоненте *Label* номера столбцов матрицы D(7, 3), содержащие положительные кратные трем числа.

5. Вывести в компоненте *ShowMessage* количество ненулевых элементов по строкам матрицы R(3, 9).

6. Рассчитать прирост населения по шести городам-миллионникам за квартал. Интерфейс программы приведен на рис. 27. Алгоритм программы приведен в Блок-схеме.

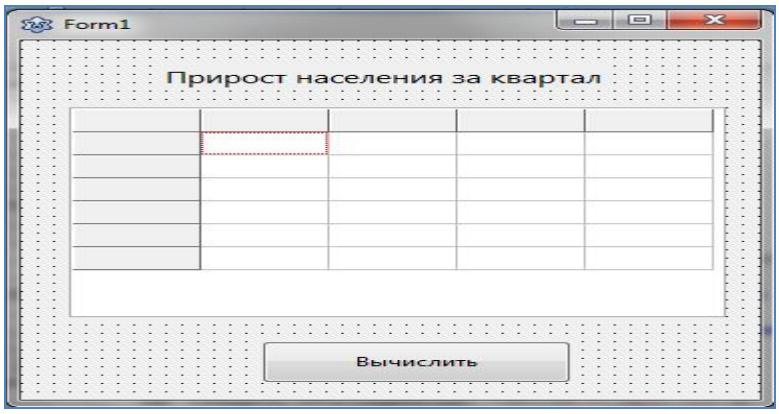


Рис. 27. Интерфейс программы *Прирост населения*

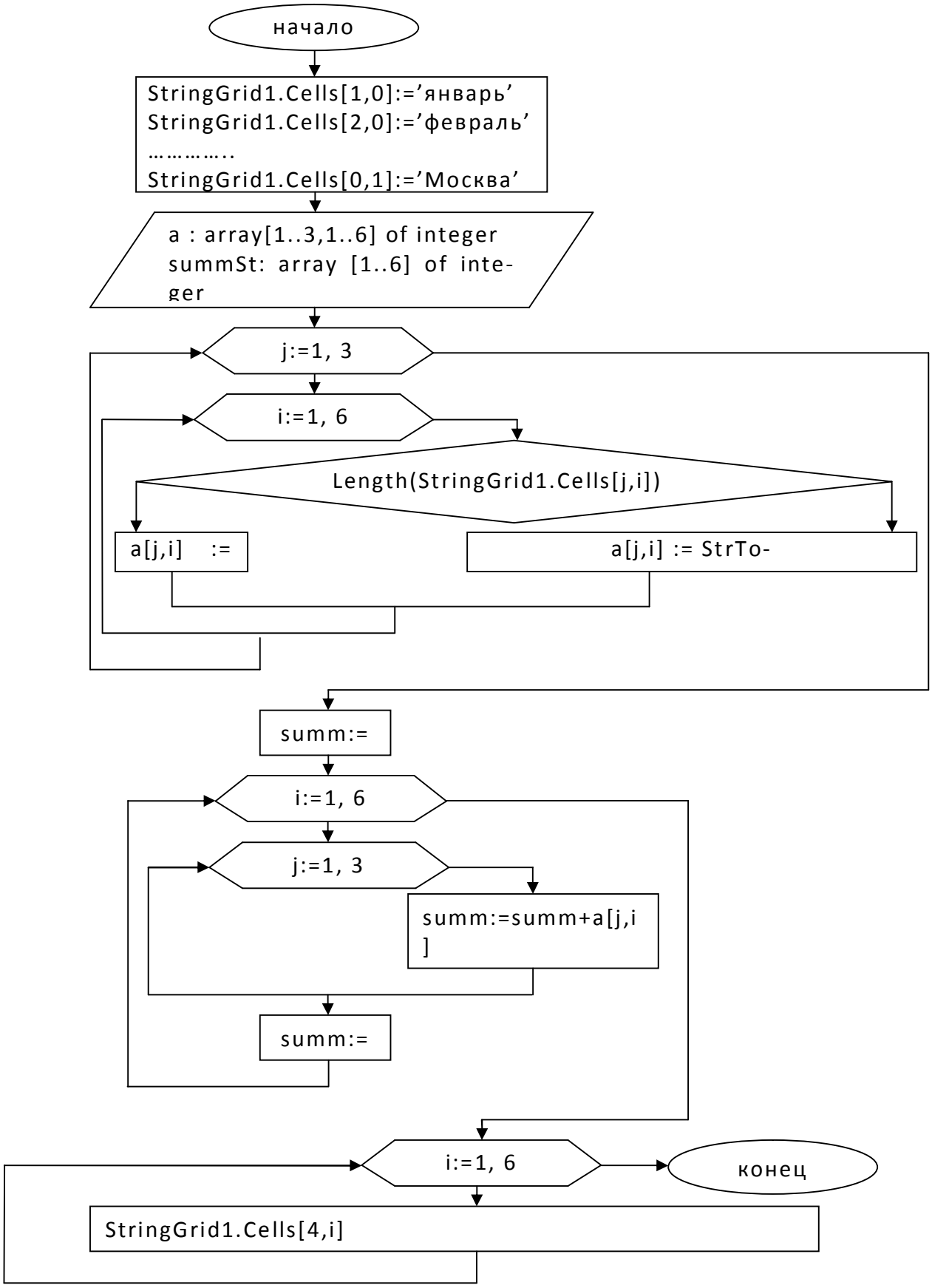
Решение:

а) заголовки строк и столбцов таблицы заполняются событием *OnCreate*для *Form1*, расчет итоговых значений выполняется событием *OnClick* для *Button1*;

б) обозначения:

a – массив целых чисел *StringGrid*;
j – столбец;
i – строка;
summ – сумма элементов по строке;
SummSt – массив рассчетных итоговых значений по всем строкам таблицы.

7. Рассчитать заработную плату 10 сотрудников бригады строителей. Известны фамилии, количество отработанных смен в месяце, стоимость смены работы, процент премии каждого работника, уральский коэффициент и подоходный налог.



Блок-схема. Алгоритм решения задачи *Прирост населения*

Лабораторная работа № 4

Цель: использование файлов.

Файл – это именованная структура данных, представляющая собой последовательность элементов данных одного типа, причем количество элементов последовательности практически не ограничено.

Как и любая структура данных (переменная, массив) программы, файл должен быть объявлен в разделе описания переменных. При объявлении файла указывается тип элементов файла. В общем виде объявление файла выглядит так:

Например,

Имя: **fileof***ТипЭлементов*

Res: **fileof**integer; *// файлцелыхчисел*

Koef: **fileof**real; *// файл вещественных чисел*

Файл, компонентами которого являются данные символьного типа, называется *символьным*, или *текстовым*.

Описание текстового файла в общем виде выглядит так:

Имя: TextFile

Здесь

- *Имя* – имя файловой переменной;
- *TextFile* – обозначение типа, показывающее, что *Имя* – это файловая переменная, представляющая текстовый файл.

Объявление файловой переменной задает только тип компонентов файла. Для того чтобы программа могла выводить данные в файл или считывать данные из файла, необходимо указать конкретный файл, т.е. связать файловую переменную с конкретным файлом (задать имя файла).

Имя файла задается вызовом процедуры **AssignFile**, связывающей файловую переменную с конкретным файлом.

Описание процедуры *AssignFile* выглядит следующим образом:

*AssignFile (***var***f, ИмяФайла:* **string***)*

Имя файла задается согласно принятым в Windows правилам, т.е. включает полный путь к файлу (имя диска, каталогов и подкаталогов).

Примеры вызова процедуры *AssignFile*:

```
AssignFile (f, 'a:\result.txt')
AssignFile (f, '\students\ivanov\korni.txt')
Fname := ('otchet.txt')
AssignFile (f, fname)
```

Вывод в текстовый файл осуществляется с помощью инструкции **write** или **writeln**. Инструкция *writeln* после вывода всех значений, указанных в списке вывода, записывает в файл символ «новая строка». В общем виде эти инструкции записываются следующим образом:

Write *(ФайловаяПеременная, СписокВывода)*
Writeln *(ФайловаяПеременная, СписокВывода)*

Здесь

- *ФайловаяПеременная* – переменная, идентифицирующая файл, в который выполняется вывод;
- *СписокВывода* – разделенные запятыми имена переменных, значения которых надо вывести в файл. Помимо имен переменных в список вывода можно включать строковые константы.

Например, если переменная *f* является переменной типа *TextFile*, то инструкция вывода значений переменных *x1* и *x2* в файл может быть такой:

Write (f, ‘Корни уравнения’, x1, x2)

Перед выводом в файл его необходимо открыть. Возможны следующие режимы для записи в файл:

1) перезаписать (запись нового файла поверх существующего или создание нового файла);

2) добавление в существующий файл.

Для того чтобы открыть файл в режиме создания нового файла или замены существующего, необходимо вызвать процедуру **Rewrite(f)**, где *f*– файловая переменная типа *TextFile*.

Для того чтобы открыть файл в режиме добавления к уже существующим данным, находящимся в этом файле, нужно вызвать процедуру **Append(f)**, где *f* – файловая переменная типа *TextFile*.

Программа может взять на себя контроль за результатом выполнения инструкции открытия файла инструкцией **IOResult** (*input-outputresult*): возвращает ноль, если операция ввода/вывода завершилась успешно, в противном случае – ошибка. Для выполнения проверки необходимо перед инструкцией вызова процедуры открытия файла поместить директиву вида **{$I-}**, которая запрещает автоматическую обработку ошибок ввода\вывода. Эта директива сообщает компилятору, что программа берет на себя контроль ошибок. После инструкции открытия файла следует поместить директиву **{$I+}**, восстанавливающую режим автоматической обработки ошибок ввода\вывода.

Задание к Лабораторной работе № 4

Задача 1. Составить программу, которая выполняет запись и добавление данных в текстовый файл. Имя файла вводится во время работы в поле *Edit1*. Нажатие командной кнопки *Записать* открывает файл в режиме создания нового или замещения существующего файла и записывает текст, находящийся в поле компонента *Memo1*. Нажатие на командную кнопку *Добавить* открывает файл, имя которого указано в поле *Edit1*, и добавляет в него содержимое поля *Memo1*.

1. Создать папку Lab41.
2. Создать в папке Lab41текстовый документ test.txt.
3. Поместить на форму компоненты (рис. 28).

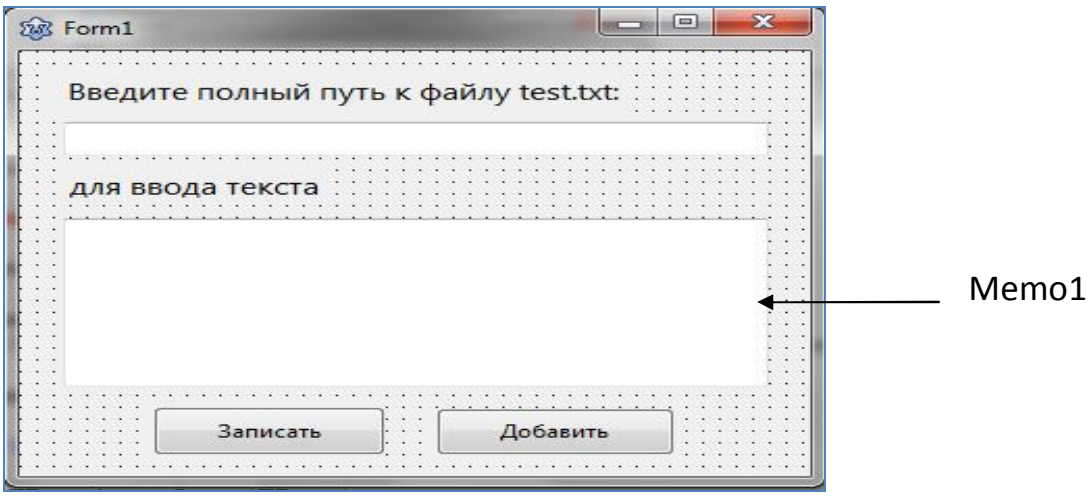


Рис. 28. Интерфейс программы *Работа с файлами*

4. Ввести текст события *OnClick* для кнопки *Записать*:

```
procedure TForm1.Button1Click(Sender: TObject);
var
f: TextFile;       // файл
fName: String[80]; // имя файла
i: integer;
begin
fName := Edit1.Text;
AssignFile(f, fName);
Rewrite(f);  // открыть для перезаписи

// запись в файл
for i:=0 to Memo1.Lines.Count-1 do // строки нумеруются с нуля
writeln(f, Memo1.Lines[i]);
CloseFile(f); // закрыть файл
MessageDlg('Данные ЗАПИСАНЫ в файл ',mtInformation,[mbOk],0);
end;
```

5. Ввести текст события *OnClick* для кнопки *Добавить*:

```
procedure TForm1.Button2Click(Sender: TObject);
var
f: TextFile;       // файл
fName: String[80]; // имяфайла
i: integer;
begin
fName := Edit1.Text;
AssignFile(f, fName);
Append(f);  // открыть для добавления

  // запись в файл
for i:=0 to Memo1.Lines.Count-1 do // строки нумеруются с нуля
writeln(f, Memo1.Lines[i]);
CloseFile(f); // закрыть файл
MessageDlg('Данные ДОБАВЛЕНЫ в файл ',mtInformation,[mbOk],0);
end;
```

6. Проверить работу программы.

Задача 2. Записать в текстовый файл введенные пользователем данные о результатах соревнований, формируя базу данных. Исходные данные вводятся в поля диалогового окна (рис. 29) и сохраняются в текстовом файле Medals.txt.

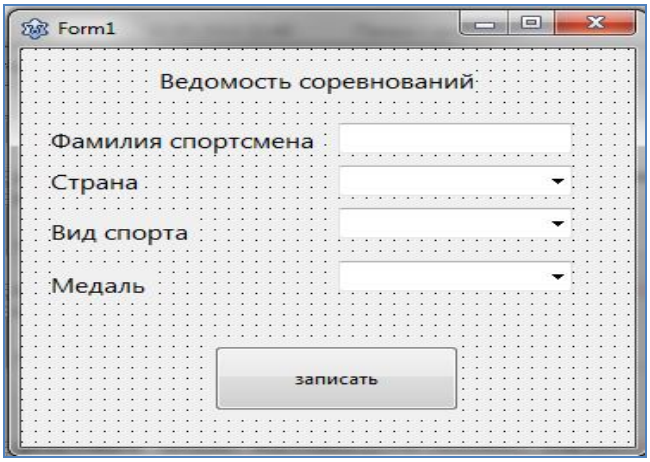


Рис. 29. Интерфейс программы *Ведомость соревнований*

1. Создать папку Lab42.
2. Создать в папке Lab42 текстовый документ Medals.txt.
3. Заполнить значения списков *ComboBox1*, *ComboBox2*, *ComboBox3* произвольными значениями.
4. Дополнить листинг программы:

```
implementation
{$R *.lfm}
const
```

```
DBName= 'F:\Lab42\Medals.txt';\\путь к текстовому файлу Medals.txt
var
f:TextFile;

{TForm1 }

procedure TForm1.FormActivate(Sender: TObject);
begin
AssignFile(f,dbName);
    {$I-}
Append(f);

\\ сделать недоступными поля ввода в случае, когда файл БД не найден
If IOResult<>0 then
begin
Edit1.Enabled:=False;
 ComboBox1.Enabled:=False;
 ComboBox2.Enabled:=False;
ComboBox3.Enabled:=False;
ShowMessage('Ошибка! ФайлБДненайден ' + DBName);
end;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
Append(f);
if (Length(Edit1.Text)=0) then
ShowMessage('поле СПОРТСМЕН должно быть заполнено!')
elseWriteLn(f,  Edit1.Text,  ' ',  ComboBox1.Text,  '' ,  ComboBox2.Text,  ' ' ,
ComboBox3.Text);
\\ очистить поля
Edit1.Text:=' ';
ComboBox1.Text:=' ';
   ComboBox2.Text:=' ';
   ComboBox3.Text:=' ';
CloseFile(f);
end;
```

5. Проверить работу программы.

Дополнительные задания

Добавить на форму *Ведомость соревнований* поле *Дистанция* (*ComboBox4*) так, чтобы в текстовом файле Medals.txt отражалась дополнительная информация о пройденной спортсменом дистанции.

Лабораторная работа № 5

Цель: использование динамических структур данных, создание списков.

Обычно переменная хранит некоторые данные. Помимо обычных переменных существуют переменные, которые ссылаются на другие переменные. Такие переменные называются *указателями*.

Указатель – это переменная, значением которой является адрес другой переменной или структуры данных. Графически указатель можно изобразить так:

Указатель, как и любая переменная программы, должен быть объявлен в разделе объявления переменных. В общем виде объявление указателя выглядит следующим образом:

Имя: ^Тип

Здесь

- *Имя* – имя переменной-указателя;
- *Тип* – тип переменной, на которую указывает переменная-указатель;
- значок ^ показывает, что объявляемая переменная является указателем.

Например:

```
p1: ^integer;
p2: ^real;
```

В приведенном примере переменная *p1* – это указатель на переменную типа *integer*, а *p2* – указатель на переменную типа *real*.

В начале работы программы переменная-указатель “ни на что не указывает”, поэтому значение указателя равно *NIL*. Зарезервированное слово *NIL* соответствует значению указателя, который ни на что не указывает.

Указателю можно присвоить значение – адрес переменной соответствующего типа (оператор @).

Например, инструкция, после выполнения которой переменная p будет содержать адрес переменной n:

```
p:=@n
```

Динамической переменной называется переменная, память для которой выделяется во время работы компьютера. Выделение памяти для динамической переменной осуществляется вызовом процедуры **New**. У процедуры *New* один параметр – указатель на переменную того типа, память для которой надо выделить. У динамической переменной нет имени, поэтому обратиться к ней можно только с помощью указателя.

Например, если *p* является указателем на тип *real*, то в результате выполнения процедуры *new(p)* будет выделена память для переменной типа

real и переменная-указатель *p* будет содержать адрес памяти, выделенной для этой переменной.

Процедура, использующая динамические переменные, перед завершением своей работы должна освободить занимаемую этими переменными память. Для освобождения памяти, занимаемой динамической переменной, используется процедура **Dispose**, которая имеет один параметр – указатель на динамическую переменную.

Например, если *p* – указатель на динамическую переменную, память для которой выделена инструкцией *new(p),* то инструкция *dispose(p)* освобождает занимаемую динамической переменной память.

Списки

Указатели динамических переменных позволяют создавать сложные динамические структуры данных, такие, как *списки* и *деревья*.

Каждый элемент списка представляет собой запись, состоящую из двух частей: первая часть информационная, вторая часть отвечает за связь со следующим и, возможно, с предыдущим элементом списка.

Для того, чтобы программа могла использовать список, надо определить тип компонентов списка и переменную – указатель на первый элемент списка. Ниже приведен пример объявления компонента списка студентов.

```
Type
   TPStudent = ^TStudent;  //указатель на переменную типа TStudent
// описание типа элементов списка
TStudent = record
   Surname: string[20];    // фамилия
   Name:    string[20];    //имя
   Group:   integer;       //номер группы
   Address: string[60];    //домашний адрес
   Next:    TPStudent;     //указатель на следующий элемент списка
End;
Var
  Head: TPStudent;   //указатель на первый элемент списка
```

Задание к Лабораторной работе № 5

Задача. Написать программу, формирующую список элементов, добавляя фамилию и имя студента в начало списка. Данные вводятся в поля *Edit1*и *Edit2* диалогового окна и добавляются в список нажатием кнопки *Добавить*. Вывод списка осуществляется нажатием кнопки *Показать*. Удаление элемента списка выполняется нажатием кнопки *Удалить*. Если узла, который хочет удалить пользователь из списка, нет, то программа выводит сообщение об ошибке. Для доступа к элементам списка используется указатель *curr*.

1. Создать папку Lab5.
2. Добавить компоненты (рис. 30).

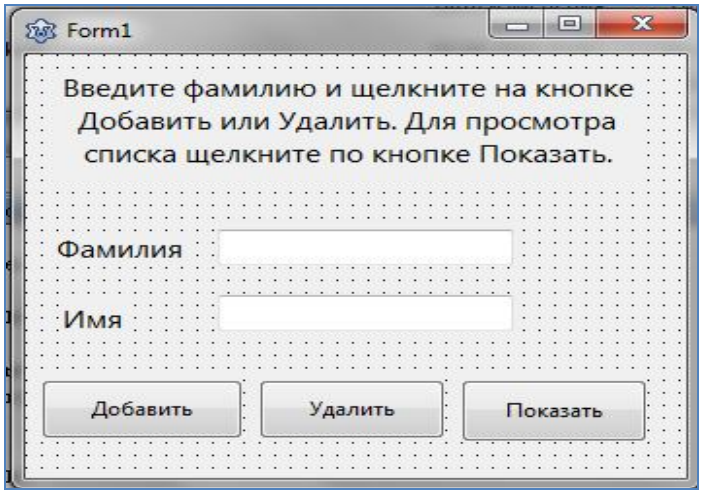


Рис. 30. Интерфейс программы *Динамический список*

3. Дополнить листинг программы:

……..
implementation

```
{$R *.lfm}
type
TPStudent=^TStudent; \\ указатель на тип TPStudent
TStudent = record
fam:string[20];   \\ фамилия
name:string[20];   \\ имя
next:TPStudent;   \\ следующий элемент списка
end;
var
head: TPStudent;\\ начало (голова) списка
{ TForm1 }

\\ добавить элемент в список
procedure TForm1.Button1Click(Sender: TObject);
var
nov:TPStudent;\\ новый элемент списка
begin
new(nov);\\ создание нового элемента списка
nov^.fam:=Edit1.Text;\\ фамилия
nov^.name:=Edit2.Text;\\ имя
nov^.next:=head;
head:=nov;
 Edit1.Text:='';
    Edit2.Text:='';
    Edit1.SetFocus;
end;

\\ отобразить список
procedure TForm1.Button3Click(Sender: TObject);
```

```
var
tek:TPStudent; \\ текущий элемент списка
n:integer; \\ количество элементов списка
st: string; \\строковое представление списка
begin
n:=0;
st:='';
tek:=head;
whiletek<>nil do
begin
n:=n+1;
st:=st+tek^.fam+' '+ tek^.name+#13;
tek:=tek^.next;
end;
if n<>0
thenShowMessage('Список:'+#13+st) else ShowMessage('Список пуст!');
end;

\\ удалить элемент из списка
procedure TForm1.Button2Click(Sender: TObject);
var
tek:TPStudent;   \\ текущий проверяемый узел
pre:TPStudent;  \\ предыдущий узел
del:boolean;  \\ узел, который надо удалить
begin
if head=nil then
begin
MessageDlg('Список пуст!', mtError, [mbOk],0);
exit;
end;
tek:=head; \\ текущий узел -  первый узел
pre:=nil;\\ предыдущего узла нет
del:=false;

\\ найти узел, который надо удалить
while (tek<>nil) and (not del) do
begin
if (tek^.fam=Edit1.Text) and (tek^.name=Edit2.text)
then del:=true  \\ нужный узел найден
else\\ к следующему узлу
begin
pre:=tek;
tek:=tek^.next;
end;
end;
if del then
begin
ifMessageDlg('Узел будет удален из списка!', mtWarning,
[mbOk,mbCancel],0)=mrYes
then exit;
```

```
if pre=nil
then head:=tek^.next else pre^.next:=tek^.next;
Dispose(tek);
MessageDlg('Узел найден и будет удален из списка!'+#13+'имя-
'+Edit1.Text+'фамилия-'+Edit2.Text, mtInformation, [mbOk],0);
end
else\\ узла, который надо удалить, в списке нет
MessageDlg('Узел '+'имя-'+Edit1.Text+'фамилия-'+Edit2.Text+#13+'не
найден!', mtError, [mbOk],0);
edit1.text:='';
  edit2.text:='';
  edit1.SetFocus;
end;

\\ начало работы программы
procedure TForm1.FormActivate(Sender: TObject);
begin
head:=nil;
end;
```

4. Проверить работу приложения.

Лабораторная работа № 6

Цель: работа с базами данных в *Lazarus*.

Базы данных в *Lazarus* основывают свою работу на базовом классе **TDataSet**. Этот класс представляет таблицу или результат запроса в приложении. Как и многие другие базовые классы, *TDataSet* нельзя использовать в своём приложении непосредственно, а можно использовать только его потомков, которых довольно много. Потомки обеспечивают доступ к различным видам баз данных (локальный *dBase* или текстовые файлы), а также к серверам баз данных (*PostgreSQL*, *Firebird*, *MySQL* и т.п). Некоторые потомки *TDataSet* связываются непосредственно с таблицами базы данных, в то время как другие используют дополнительные компоненты или библиотеки.

В общем случае, первым шагом необходимо выбрать соответствующего потомка *TDataSet* для выбранной базы данных, создать подключение, выбрать таблицу или создать запрос и открыть его. Когда набор открыт, создается ряд компонентов по одному для каждого поля или столбца таблицы (запроса). Каждый компонент поля – это потомок *TField*, соответствующий специфическому типу данных этого поля, например, *TStringField*.

Для работы с базами данных используются компоненты, расположенные на вкладках *DataAccess* и *DataControls* с приставкой *DB*. Кроме свойства *DataSource*, у этих компонентов необходимо установить в свойстве *DataField* имя связываемого поля. В табл. 18 приведен список часто используемых при работе с базами данных компонентов.

Таблица 18

Список компонентов для работы с базами данных

Компонент	Назначение
DBEdit (DBText)	текстовое поле, доступное (недоступное) для редактирования
DBMemo	многострочное текстовое поле, доступное для редактирования
DBImage	рисунок, который хранится в базе данных BLOB-поле
DBListDBComboBox	заносит в базу данных значения из предопределенного списка, который хранится в свойстве Items
DBCheckBox	булево поле, доступное для редактирования
DBRadioGroup	отображает значения в виде группы радиокнопок
DBCalendar	отображает\редактирует поля с датой в виде календаря
DBNavigator	добавление\удаление записи, переход по записям

Задание к Лабораторной работе № 6

Задача. Разработать интерфейс для удобного использования текстового файла, содержащего контактную информацию о людях.

1. Создать папку Lab6.

2. В папке Lab6 создать текстовый файл Baza.txt, который заполнить произвольными данными (слова разделять кнопкой <Tab>): Фамилия, Имя, Отчество, e-mail, Телефон.

3. Добавить на форму с вкладки *DataControls* компоненты *DBEdit*, *DBNavigator*, с вкладки *DataAccess* - *SdfDataSet*, *DataSource*, как на рис. 31.

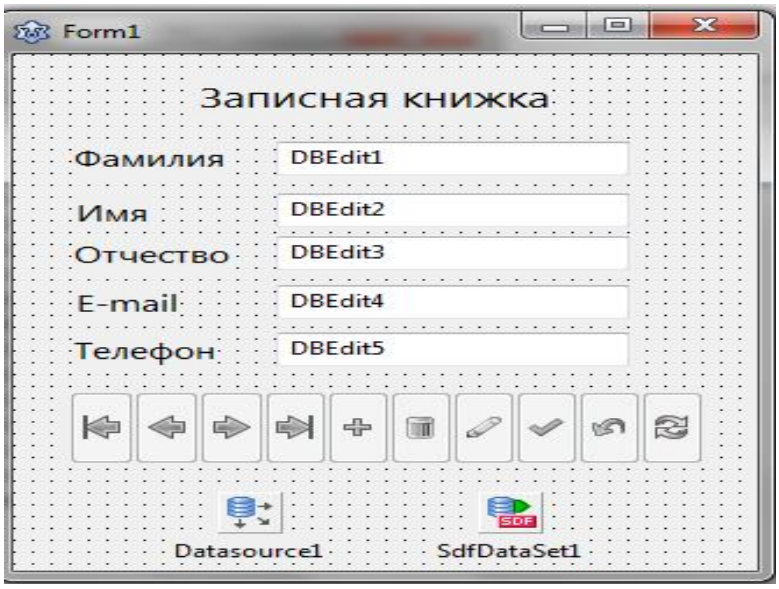


Рис. 31. Интерфейс программы *Записная книжка*

4. Для компонента *SdfDataSet1* указать значения свойств:
   - *FileName* – полный путь к файлу Baza.txt;
   - *AutoCalcFields* – False;
   - *Schema* (определяет набор полей) – добавить имена полей: фамилия, имя, отчество, e-mail, телефон;
   - *Delimiter* - #9;
   - *Active* – True.

5. Для компонента *DataSource1* свойству *DataSet*задать значение *SdfDataSet1*.

6. Связать компоненты *DBEdit1..DBEdit5* с полями базы данных по свойствам:
   - *DataSourse* – задать значение *DataSource1*;
   - *DataField* – имя поля базы данных.

7. Связать компонент *DBNavigator* с *DataSource1*.

8. Проверить работу приложения.

Дополнительные задания

1. Использовать компонент *DBGrid* для размещения контактной информации из задачи *Записная книжка* в виде таблицы.

2. Разработать интерфейс базы данных по индивидуальному заданию.

Используемая литература

1. Алексеев Е.Р., Чеснокова О.В., Кучер Т.В. Самоучитель по программированию на FreePascal и Lazarus. - Донецк: ДонНТУ, Технопарк ДонНТУ УНИТЕХ, 2009. – 503 с.

2. Культин Н.Б. Delphi в задачах и примерах. – 2-е изд., перераб. и доп. – СПб.:БХВ-Петербург, 2008. – 288 с.:ил.+CD-ROM

3. Культин Н.Б. Основы программирования в Delphi. – СПб.: БХВ-Петербург, 2009. – 608 с.:ил.

Содержание