© Составление: Выграненко М.В., vygranenko@mail.ru Апрель 2012

Система программирования Lazarus

Система объектно-ориентированного программирования Lazarus является кроссплатформенной свободно распространяемой системой, аналогичной Delphi.

Lazarus — это инструмент разработки программ, использующий интерфейс IDE. IDE расшифровывается как Integrated Development Environment (Интегрированная среда разработки). Эта среда облегчает разработку программ. Например, если вы добавляете на форму кнопку, система программирования автоматически создает код для этой кнопки. (Конечно, не полностью.) Система программирования облегчает чтение кода, автоматически раскрашивая команды кода в зависимости от их назначения. Lazarus помогает упорядочивать код, храня его в нескольких отдельных файлах. Это позволяет разделить программу на функциональные блоки. Например, код каждой формы можно хранить в отдельном файле. Система также позволяет легко компилировать и запускать программы. С помощью утилит отладки, входящих в неё, можно искать ошибки в программе и отслеживать ее выполнение. Система объектно-ориентированного программирования Lazarus представляет собой универсальный инструмент, с помощью которого можно создавать разнообразные приложения.

Дистрибутив доступен для «скачивания» с сайта компании:

http://sourceforge.net/projects/lazarus/files/.

Система программирования **Lazarus** является системой "Объектно-ориентированного программирования" (ООП), в основе которой заложено понятие "**объект**", объединяющее в себе определённые **свойства и методы** (действия объектов и над объектами). Такое объединение свойств и методов в объекте называется **инкапсуляцией**.

Программные объекты обладают **свойствами**, могут использовать **методы** и реагируют на **события**.

Свойства объектов (Properties). Каждый объект обладает определённым набором свойств. Например свойства кнопки - ширина, высота, текст надписи, параметры шрифта и т.д. Первоначальные значения свойств задаются в режиме конструирования. Но в программном коде можно задать изменение значения свойства. Вид такой команды: Имя_объекта.Свойство := Значение Свойства

Правило записи свойства объекта через точку (в левой части команды) называют точечной нотацией.

Методы объектов (Methods). С каждым объектом можно произвести какие-то действия: например, показать, скрыть, масштабировать, поместить в фокус и т.п. Многие методы имеют аргументы, позволяющие задавать параметры этих действий.

События (Events). Событие - это действие, распознаваемое объектом (щелчок мыши, нажатие клавиши и т.п.) Реакция на событие - вызов процедуры, которая может менять свойства объекта, вызывать его методы и т.п.

Классы объектов являются "шаблонами", определяющими наборы свойств, методов и событий, по которым создаются экземпляры класса. Классы могут иметь взаимоотношения друг с другом, например:

- **наследование** - когда экземпляр класса наследует от базового класса все данные и код реализации;
- **включение** - когда один класс содержит в себе переменную, указывающую на другой класс, и использует её для вызова членов этого класса.

Формы и элементы управления

Когда Вы создаете приложение Windows в Lazarus, система автоматически добавляет в Ваш проект файл формы Unit1.lfm.

В форме размещают все компоненты, которые и будут составлять внутреннее содержание выполняемой программы. Все эти компоненты приведены в окне Палитра компонентов. **Форма**, собственно, и является тем пользовательским интерфейсом, который будет отображаться на экране во время выполнения приложения. Форму можно ассоциировать с создаваемым окном, которое по умолчанию называется Form1, и ее можно редактировать в процессе разработки, т.е. изменять размер, цвет, размещать на ней компоненты и удалять их из формы и т.д. Для внесения всех этих изменений существует связанное с формой окно **Инспектор объектов**.

Вкладки окна **Инспектор объектов** автоматически создаются для каждого вновь созданного в форме объекта и являются основным инструментом для настройки объектов. Необходимая вкладка выбирается из списка объектов, находящегося в верхней части окна, по имени объекта.

После того как необходимый объект выбран, можно не только изменять и настраивать все его свойства, но и создавать обработчики событий для данного объекта, которые будут определенным образом реагировать на такие события, как, например, щелчок кнопки мыши, перемещение мыши или нажатие клавиш клавиатуры.

При этом для каждого объекта в инспекторе объектов существуют две вкладки: Свойства и События.

На форме можно располагать любые объекты, в том числе и элементы управления, например кнопки, надписи, текстовые поля, выпадающие списки и т. д. Эти элементы управления позволяют пользователю взаимодействовать с программой.

У каждого элемента управления есть **событие по умолчанию**. Например:

- Для кнопки событие по умолчанию — Click (щелчок по кнопке).
- Для текстового поля событие по умолчанию — Change (изменение текста).
- Для позиции переключателя событие по умолчанию — Change (включение/ выключение).
- Для флажка событие по умолчанию — Change (установка/ сброс флажка).

Кроме того, у каждого элемента управления есть множество событий, помимо события по умолчанию. Например, у текстового поля есть такие события:

- Click (щелчок по текстовому полю);
- Dblclick (двойной щелчок по текстовому полю);
- MouseEnter (попадание курсора мыши на текстовое поле);
- MouseLeave (уход курсора мыши с текстового поля);
- и др.

Чтобы увидеть все события, которые есть у элемента, откройте вкладку **События** в **Инспекторе объектов**.

События **генерируются** в результате действий пользователя. Например, события генерируются, когда пользователь нажимает кнопку, выбирает пункт в выпадающем списке или изменяет текст в текстовом поле. Щелчки или двойные щелчки по элементам управления, перемещение курсора на эти элементы управления или с них тоже генерируют события. Когда происходят эти события, исполняется код, связанный с ними. Процесс событие-отклик-событие-отклик называется событийным управлением, он лежит в основе работы всех современных графических программ.

Кодирование на языке Lazarus

Редактор кода открывается двойным щелчком на программируемом объекте. Одновременно создаётся событийная процедура (обработчик события по умолчанию) для этого объекта. В коде автоматически прописывается начало и конец этой процедуры, а действия, которые должны быть выполнены в результате этой процедуры, надо, конечно, вписывать в код самому. Редактор кода представляет собой полнофункциональный текстовый редактор, с помощью которого можно просматривать и редактировать исходный код программы. По умолчанию с окном редактора кода состыковано окно сообщений. Оно появляется автоматически, если в процессе компиляции программы были сгенерированы сообщения об ошибках или предупреждающие сообщения

Современные языки программирования, позволяют добавлять в код программ **комментарии**. Комментарии не компилируются вместе с программой и не исполняются при ее выполнении. Они позволяют добавлять в программу замечания, поясняющие ее работу и назначение. Хороший программист всегда выделит время на документирование и объяснение написанного им кода.

В Lazarus комментарий в строке начинается с символа двойной слеш (//).Можно помечать строки кода как строки комментариев, чтобы эти строки не выполнялись. Этот прием называется «закомментировать строку кода». Если в вашей программе есть ошибка, можете по очереди закомментировать отдельные строки или целые блоки кода, пока ошибка не перестанет появляться. Ошибка, скорее всего, будет находиться в строке, которую вы закомментировали последней, прежде чем ошибка исчезла.

Комментарии выделяются в программах синим цветом. Это позволяет легко замечать их в коде.

Во многих случаях Lazarus автоматически добавляет пропуски и отступы, чтобы код было легче читать. Если хотите, Вы тоже можете добавлять дополнительные пропуски и отступы. Они облегчают чтение и понимание кода и комментариев. Lazarus не обращает внимания на пустые строки и отступы. Они игнорируются при компиляции программы.

Часто Вашим программам понадобится вести диалог с пользователем. Для этого Lazarus поддерживает набор стандартных окон, например **окон сообщений** (MessageBox). Синтаксис использования окна сообщения таков: *ShowMessage ('Текст сообщения').* Обратите внимание, что текст сообщения должен быть помещен в апострофы.

Запись инструкций программы

Одну инструкцию от другой отделяют точкой с запятой или, другими словами, в конце каждой инструкции ставят точку с запятой. Хотя в одной строке программы можно записать несколько инструкций, как правило, каждую инструкцию программы записывают в отдельной строке. Некоторые инструкции (if, case, repeat, while и др.) принято записывать в несколько строк, используя для выделения структуры инструкции отступы.

Длинные выражения тоже могут быть записаны в несколько строк. Разорвать выражение и перенести оставшуюся часть на следующую строку можно практически в любом месте. Нельзя разрывать имена переменных, числовые и строковые константы, а также составные операторы, например, оператор присваивания.

Еще один момент, на который следует обратить внимание. Компилятор игнорирует "лишние" пробелы и пустые строки. Так, он игнорирует все пробелы в начале строки. Кстати, это и позволяет записывать инструкции с отступами. Не требуются пробелы при записи арифметических и логических выражений (условий), списков параметров. Однако при их использовании программа легче воспринимается.

Для облегчения понимания логики работы программы нужно использовать поясняющий текст - комментарии. В общем случае комментарии заключают в фигурные скобки. Открывающая скобка помечает начало комментария, закрывающая - конец. Если комментарий однострочный или находится после инструкции, то перед комментарием ставят две наклонные черты.

Стиль программирования

Работая над программой, программист, особенно начинающий, должен хорошо представлять, что программа, которую он разрабатывает, предназначена, с одной стороны, для пользователя, с другой - для самого программиста. Для того, чтобы работа была эффективной, программа должна быть легко читаемой, ее структура должна соответствовать структуре и алгоритму решаемой задачи. Для этого надо следовать правилам хорошего стиля программирования:

- использование комментариев;
- использование несущих смысловую нагрузку имен переменных, процедур и функций;
- использование отступов;
- использование пустых строк.

Следование правилам хорошего стиля программирования значительно уменьшает вероятность появления ошибок на этапе набора текста, облегчает процессы отладки и внесения изменений. Хорошая программа должна быть прежде всего надежной (контролировать исходные данные, проверять результат выполнения операций) и дружественной (иметь хорошо спроектированные диалоговые окна, справочную систему, разумное и предсказуемое поведение программы) по отношению к пользователю.

Проект в lazarus

Приложение собирается из многих элементов: форм, программных модулей, внешних библиотек, картинок, пиктограмм и др. Каждый элемент размещается в отдельном файле и имеет строго определенное назначение. Набор всех файлов, необходимых для создания приложения, называется **проектом**. Компилятор последовательно обрабатывает файлы проекта и строит из них выполняемый файл. Основные файлы проекта можно разделить на несколько типов:

Главный файл проекта — текстовый файл с расширением LPR, содержащий главный программный блок. Файл проекта подключает все используемые программные модули и содержит операторы для запуска приложения. Этот файл среда Lazarus создает и контролирует сама. Файл проекта представляет собой программу, написанную на языке Object Pascal и предназначенную для обработки компилятором.

Файлы описания форм — текстовые файлы с расширением LFM, описывающие формы с компонентами. В этих файлах запоминаются начальные значения свойств, установленные вами в окне свойств.

Файлы программных модулей — текстовые файлы с расширением PAS, содержащие исходные программные коды на языке Lazarus. В этих файлах вы пишите методы обработки событий, генерируемых формами и компонентами.

Модуль — автономно компилируемая программная единица, включающая в себя различные компоненты раздела описаний (типы, константы, переменные, процедуры и функции) и, возможно некоторые исполняемые операторы инициирующей части.
Любой модуль имеет следующую структуру:

Заголовок модуля	unit <имя модуля>;
Директивы компилятора	{$<директивы>}
Интерфейсная часть	interface
Подключение модулей	uses <имя>, ..., <имя>;
Константы	const ... ;
Типы данных	type ... ;
Переменные	var ... ;
Заголовки процедур	procedure <имя> (<параметры>);
Заголовки функций	function <имя> (<параметры>): <тип>;
Часть реализации	implementation
Подключение модулей	uses <имя>, ..., <имя>;
Константы	const ... ;
Типы данных	type ... ;
Переменные	var ... ;
Реализация процедур	procedure <имя>; begin ... end;
Реализация функций	function <имя>; begin ... end;
Код инициализации	initialization <операторы>
Код завершения	finalization <операторы>
	end.

Заголовок	открывается зарезервированным словом UNIT, за которым следует имя модуля и точка с запятой. Имя модуля служит для его связи с другими модулями и основной программой. Эта связь устанавливается специальным предложением USES <список модулей>
Секция интерфейсных объявлений	открывается зарезервированным словом INTERFACE. В этой части содержатся объявления всех глобальных объектов модуля (типов, констант, переменных и подпрограмм), которые должны стать доступными основной программе и (или) другим модулям.
Секция реализаций	открывается словом IMPLEMENTATION, содержит описание подпрограмм. В ней могут появляться локальные для модуля объекты.
Секция инициации	открывается словом BEGIN.
Терминатор	модуля, как и терминатор программы - END с точкой .

Начиная работу со средой Lazarus, Вы должны:

1. Уяснить задачу, которую собираетесь решать на компьютере;
2. Нарисовать на бумаге все то, что предполагаете увидеть на экране в процессе решения. Это может быть один или несколько рисунков. Если задача сложная, ее следует разбить на этапы и для каждого этапа сделать отдельный рисунок;
3. Написать сценарий работы будущей программы. Местом развертывания действия является экран, а зритель не просто смотрит, но и участвует в "спектакле". В сценарии должно быть учтено все: что выводится на экран вначале, что делается потом, как программа завершается и т.д. Декорациями "спектакля" служат сделанные в пункте 2 рисунки.

Типы данных

Программа может оперировать **данными** различных типов: целыми и дробными числами, символами, строками символов, логическими величинами.

Целый тип

Язык Lazarus поддерживает несколько целых типов данных. Типы различаются между собой диапазоном допустимых значений, количеством значащих цифр и количеством байтов, необходимых для хранения данных в памяти компьютера. В таблице приведены некоторые из них:

Тип	Диапазон	Формат
Integer	От -2 147 483 648 до 2 147 483 647	32 Бита
Shortint	От -128 до 127	8 Бит
Smallint	От -32768 до 32767	16 Бит
Longint	От -2 147 483 648 до 2 147 483 647	32 Бита
Byte	От 0 до 255	8 Бит, беззнаковый
Word	От 0 до 65 535	16 Бит, беззнаковый

Вещественный тип

Язык Lazarus поддерживает несколько вещественных типов данных. Некоторые из них:

Тип	Диапазон	Значащих цифр	Байтов
Real	$2.9* 10^{-39}$-$1.7*10^{38}$	11-12	6
Single	$1.5 * 10^{-45}$-$3.4* 10^{38}$	7-8	4
Double	$5.0*10^{-324}$ -$1.7*10^{308}$	15-16	8
Extended	$3.6*10^{-4951}$ -$1.1 *10^{4932}$	19-20	10

Строковый тип

Значением строкового типа является последовательность символов с динамически изменяемой длиною:

Тип	Максимальная длина	Байтов
Shortstring	255 символов	От 2 до 256
WideString	2^{30} символов	От4 до 2Гбайт
Ansistring	2^{31} символов	От4 до 2Гбайт

Логический тип

Логическая величина может принимать одно из двух значений True (истина) или False (ложь). В языке Lazarus логические величины относят к типу Boolean.

Данные могут быть как константами (постоянными), так и переменными величинами.

Константы и переменные на языке Lazarus

Переменные

Переменная - это область памяти, в которой находятся данные, которыми оперирует программа. Когда программа манипулирует с данными, она, фактически, оперирует содержимым ячеек памяти, т.е. переменными. Чтобы программа могла обратиться к переменной (области памяти), например, для того, чтобы получить исходные данные для

расчета по формуле или сохранить результат, переменная должна иметь имя. Имя переменной придумывает программист. В качестве имени переменной можно использовать последовательность из букв латинского алфавита, цифр и некоторых специальных символов. Первым символом в имени переменной должна быть буква. Пробел в имени переменной использовать нельзя.

Следует обратить внимание на то, что компилятор языка Lazarus не различает прописные и строчные буквы в именах переменных. Желательно, чтобы имя переменной было логически связано с ее назначением.

В языке Lazarus каждая переменная перед использованием должна быть объявлена. С помощью объявления устанавливается не только факт существования переменной, но и задается ее тип, чем указывается и диапазон допустимых значений.

В общем виде инструкция объявления переменной выглядит так:

Имя : тип;

где:
имя - имя переменной;
тип - тип данных, для хранения которых предназначена переменная.

В тексте программы объявление каждой переменной, как правило, помещают на отдельной строке. Если в программе имеется несколько переменных, относящихся к одному типу, то имена этих переменных можно перечислить в одной строке через запятую, а тип переменных указать после имени последней переменной через двоеточие.

Область действия идентификаторов

При программировании необходимо соблюдать ряд правил, регламентирующих использование идентификаторов:

- каждый идентификатор должен быть описан перед тем, как он будет использован;
- областью действия идентификатора является блок, в котором он описан;
- все идентификаторы в блоке должны быть уникальными, т.е. не повторяться;
- один и тот же идентификатор может быть по-разному определен в каждом отдельном блоке, при этом блоки могут быть вложенными;
- если один и тот же идентификатор определен в нескольких вложенных блоках, то в пределах вложенного блока действует вложенное описание;
- все глобальные описания подключенного модуля видны программе (подключающему модулю), как если бы они были сделаны в точке подключения;
- если подключаются несколько модулей, в которых по-разному определен один и тот же идентификатор, то определение, сделанное в последнем подключенном модуле перекрывает все остальные;
- если один и тот же идентификатор определен и в подключенном модуле, и в программе (подключающем модуле), то первый игнорируется, а используется идентификатор, определенный в программе (подключающем модуле). Доступ к идентификатору подключенного модуля возможен с помощью уточненного имени. Уточненное имя формируется из имени модуля и записанного через точку идентификатора.

Константы

В языке Lazarus существует два вида констант: обычные и именованные.

Обычная константа - это целое или дробное число, строка символов или отдельный символ, логическое значение.

Числовые константы

В тексте программы числовые константы записываются обычным образом, т. е. так же, как числа, например, при решении математических задач. При записи дробных чисел для разделения целой и дробных частей используется точка. Если константа отрицательная, то непосредственно перед первой цифрой ставится знак минус.

Дробные константы могут изображаться в виде числа с плавающей точкой. Представление в виде числа с плавающей точкой основано на том, что любое число может быть записано в алгебраической форме как произведение числа, меньшего 10, которое называется мантиссой, и степени десятки, именуемой порядком.

Строковые и символьные константы

Строковые и символьные константы заключаются в апострофы.

Логические константы

Логическое высказывание (выражение) может быть либо истинно, либо ложно. Истине соответствует константа True, значению "ложь" - константа False.

Именованная константа

Именованная константа - это имя (идентификатор), которое в программе используется вместо самой константы.

Именованная константа, как и переменная, перед использованием должна быть объявлена. В общем виде инструкция объявления именованной константы выглядит следующим образом:

константа = значение;

где:
константа - имя константы;
значение - значение константы.

Именованные константы объявляются в программе в разделе объявления констант, который начинается словом const.

После объявления именованной константы в программе вместо самой константы можно использовать ее имя. В отличие от переменной, при объявлении константы тип явно не указывают. Тип константы определяется ее видом.

Инструкция присваивания

Инструкция присваивания является основной вычислительной инструкцией. Если в программе надо выполнить вычисление, то нужно использовать инструкцию присваивания. В результате выполнения инструкции присваивания значение переменной меняется, ей присваивается значение. В общем виде инструкция присваивания выглядит так:

Имя : = Выражение;

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

Выражение

Выражение состоит из операндов и операторов. Операторы находятся между операндами и обозначают действия, которые выполняются над операндами. В качестве операндов выражения можно использовать: переменную, константу, функцию или другое выражение.

.

Оператор	Действие	Результат
+	сложение	сумма
-	вычитание	разность
*	умножение	произведение
/	деление	частное
MOD	Вычисление остатка от деления	остаток от деления одного числа на другое
DIV	Деление нацело	целая часть результата деления одного числа на другое

При вычислении значений выражений следует учитывать, что операторы имеют разный приоритет. Приоритет операторов влияет на порядок их выполнения. При вычислении значения выражения в первую очередь выполняются операторы с более высоким приоритетом. Если приоритет операторов в выражении одинаковый, то сначала выполняется тот оператор, который находится левее.

Для задания нужного порядка выполнения операций в выражении можно использовать скобки.

Выражение, заключенное в скобки, трактуется как один операнд. Это означает, что операции над операндами в скобках будут выполняться в обычном порядке, но раньше, чем операции над операндами, находящимися за скобками. При записи выражений, содержащих скобки, должна соблюдаться парность скобок, т. е. число открывающих скобок должно быть

равно числу закрывающих скобок. Нарушение парности скобок - наиболее распространенная ошибка при записи выражений.

Тип выражения

Тип выражения определяется типом операндов, входящих в выражение, и зависит от операций, выполняемых над ними. Например, если оба операнда, над которыми выполняется операция сложения, целые, то очевидно, что результат тоже является целым. А если хотя бы один из операндов дробный, то тип результата дробный, даже в том случае, если дробная часть значения выражения равна нулю.

Важно уметь определять тип выражения. При определении типа выражения следует иметь в виду, что тип константы определяется ее видом, а тип переменной задается в инструкции объявления.

Правила определения типа выражения в зависимости от типа операндов и вида оператора:

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

Выполнение инструкции присваивания

Инструкция присваивания выполняется следующим образом:

1. Сначала вычисляется значение выражения, которое находится справа от символа инструкции присваивания.
2. Затем вычисленное значение записывается в переменную, имя которой стоит слева от символа инструкции присваивания.

Инструкция присваивания считается верной, если тип выражения соответствует или может быть приведен к типу переменной, получающей значение. Во время компиляции выполняется проверка соответствия типа выражения типу переменной. Если тип выражения не соответствует типу переменной, то компилятор выводит сообщение об ошибке: Incompatible types...and..., где вместо многоточий указывается тип выражения и переменной.

Ввод и вывод данных

Ввод данных из окна 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);

где:

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 ввода появится на экране.

Следует еще раз обратить внимание на то, что значение функции inputBox строкового (string) типа. Поэтому если программе надо получить число, то введенная строка должна быть преобразована в число при помощи соответствующей функции преобразования.

Ввод из поля редактирования

Наиболее просто программа может получить исходные данные из окна ввода или из поля редактирования. Поле редактирования - это компонент Edit. Ввод данных из поля редактирования осуществляется обращением к свойству Text.

Компонент Edit1 используется для ввода исходных данных. Инструкция ввода данных в этом случае будет иметь вид: <переменная>:= StrToFloat(Edit1.Text);

Вывод результатов

Наиболее просто программа может вывести результат своей работы в окно сообщения или в поле вывода (компонент Label) диалогового окна.

Вывод в окно сообщения мы уже рассмотрели.

Вывод в окно сообщения

Окна сообщений используются для привлечения внимания пользователя. При помощи окна сообщения программа может, к примеру, проинформировать об ошибке в исходных данных или запросить подтверждение выполнения необратимой операции, например, удаления файла.

Вывести на экран окно с сообщением можно при помощи процедуры ShowMessage или функции MessageDlg.

Процедура ShowMessage выводит на экран окно с текстом и командной кнопкой ОК.

В общем виде инструкция вызова процедуры ShowMessage выглядит так:

ShowMessage('Сообщение');

где сообщение - текст, который будет выведен в окне.

Следует обратить внимание на то, что в заголовке окна сообщения, выводимого процедурой ShowMessage, указано название приложения, которое задается на вкладке Application окна Project Options. Если название приложения не задано, то в заголовке будет имя исполняемого файла.

Функция MessageDlg более универсальная. Она 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.

Вывод в поле диалогового окна

Часть диалогового окна, предназначенная для вывода информации, называется полем вывода, или полем метки. Поле вывода - это компонент Label. Содержимое поля вывода определяется значением свойства Caption. Изменить значение свойства Caption, как и большинства свойств других компонентов, можно как во время разработки формы приложения, так и во время работы программы. Для того чтобы во время работы программы изменить содержимое поля вывода, например, вывести в поле результат работы программы, нужно присвоить свойству новое значение. Свойство Caption символьного типа. Поэтому для того, чтобы во время работы программы вывести в поле метки числовое значение, нужно преобразовать число в строку, например, при помощи функции FloatToStr (для вещественный величин) или IntToStr (для целых величин).

Процедуры и функции

Стандартные функции

Для выполнения часто встречающихся вычислений и преобразований язык Lazarus предоставляет программисту ряд стандартных функций.

Значение функции связано с ее именем. Поэтому функцию можно использовать в качестве операнда выражения, например в инструкции присваивания. Так, чтобы вычислить квадратный корень, достаточно записать k:=Sqrt(n), где Sqrt - функция вычисления квадратного корня, n - переменная, которая содержит число, квадратный корень которого надо вычислить.

Функция характеризуется типом значения и типом параметров. Тип переменной, которой присваивается значение функции, должен соответствовать типу функции. Точно так же тип фактического параметра функции, т. е. параметра, который указывается при обращении к функции, должен соответствовать типу формального параметра. Если это не так, компилятор выводит сообщение об ошибке.

Математические функции

Математические функции позволяют выполнять различные вычисления:

Функция	**Значение**
Abs (n)	Абсолютное значение n
Sqrt (n)	Квадратный корень из n
Sqr (n)	Квадрат n
Sin (n)	Синус n
Cos (n)	Косинус n

Arctan (n)	Арктангенс n
Exp(n)	Экспонента n
Ln(n)	Натуральный логарифм n
Random(n)	Случайное целое число в диапазоне от 0 до n- 1

Функции преобразования

Функции преобразования наиболее часто используются в инструкциях, обеспечивающих ввод и вывод информации. Например, для того чтобы вывести в поле вывода (компонент Label) диалогового окна значение переменной типа real, необходимо преобразовать число в строку символов, изображающую данное число. Это можно сделать при помощи функции FloatToStr, которая возвращает строковое представление значения выражения, указанного в качестве параметра функции.
Например, инструкция Label1.caption := FloatToStr(x) выводит значение переменной x в поле Label1

Функция	**Значение функции**
Chr(n)	Символ, код которого равен n
IntToStr (k)	Строка, являющаяся изображением целого k
FloatToStr (n)	Строка, являющаяся изображением вещественного n
StrToInt (s)	Целое, изображением которого является строка s
StrToFloat (s)	Вещественное, изображением которого является строка s
Round (n)	Целое, полученное путем округления n по известным правилам
Trunc (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

Использование функций

Обычно функции используют в качестве операндов выражений. Параметром функции может быть константа, переменная или выражение соответствующего типа.

Пользовательские Функции и процедуры

При программировании в Lazarus работа программиста заключается в основном в разработке процедур (подпрограмм) обработки событий. При возникновении события автоматически запускается процедура обработки события, которую и должен написать программист. Задачу вызова процедуры обработки при возникновении соответствующего события берет на себя Lazarus. В языке Object Pascal основной программной единицей является подпрограмма. Различают два вида подпрограмм: процедуры и функции. Как процедура, так и функция, представляют собой последовательность инструкций, предназначенных для выполнения некоторой работы. Чтобы выполнить инструкции подпрограммы, надо вызвать эту подпрограмму. Отличие функции от процедуры заключается в том, что с именем функции связано значение, поэтому имя функции можно использовать в выражениях.

Структура процедуры

Процедура начинается с заголовка, за которым следуют:

- раздел объявления констант;
- раздел объявления типов;
- раздел объявления переменных;
- раздел инструкций.

В общем виде процедура выглядит так:

```
procedure Имя (СписокПараметров);
const    // здесь объявления констант
type     // здесь объявления типов
var      // здесь объявления переменных
begin  // здесь инструкции программы
end;
```

Заголовок процедуры состоит из слова procedure, за которым следует имя процедуры, которое используется для вызова процедуры, активизации 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.

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

После раздела объявления типов идет раздел объявления переменных, в котором объявляются (перечисляются) все переменные, используемые в программе.

Раздел объявления переменных начинается словом var.

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 (команды) процедуры.

Структура функции

Функция начинается с заголовка, за которым следуют разделы объявления констант, типов и переменных, а также раздел инструкций.

Объявление функции в общем виде выглядит следующим образом:

function Имя (СписокПараметров) : Тип;
const // начало раздела объявления констант
type // начало раздела объявления типов
var // начало раздела объявления переменных
begin // начало раздела инструкций
result := Значение; // связать с именем функции значение
end;

Заголовок функции начинается словом function, за которым следует имя функции. После имени функции в скобках приводится список параметров, за которым через двоеточие указывается тип значения, возвращаемого функцией (тип функции). Завершается заголовок функции символом "точка с запятой". За заголовком функции следуют разделы объявления констант, типов и переменных. В разделе инструкций, помимо переменных, перечисленных в разделе описания переменных, можно использовать переменную result. По завершении выполнения инструкций функции значение этой переменной становится значением функции. Поэтому среди инструкций функции обязательно должна быть инструкция, присваивающая переменной result значение. Как правило, эта инструкция является последней исполняемой инструкцией функции.

Объекты, классы объектов. Инкапсуляция, наследование, полиморфизм

Объекты — это крупнейшее достижение в современной технологии программирования. Они позволяют строить программу не из сложных процедур и функций, а из кирпичиков-объектов, заранее наделенных нужными свойствами. Внутренняя сложность объектов скрыта от программиста.

Формула объекта

Рассмотрение данных в неразрывной связи с методами их обработки позволило вывести *формулу объекта*:

Объект = Данные + Операции

На основании этой формулы была разработана методология *объектно-ориентированного программирования (ООП)*.

Природа объекта

В общем случае каждый объект "помнит" необходимую информацию, "умеет" выполнять некоторый набор действий и характеризуется набором свойств. То, что объект "помнит", хранится в его полях. То, что объект "умеет делать", реализуется в виде его внутренних процедур и функций, называемых *методами*. *Свойства* объектов аналогичны свойствам, которые мы наблюдаем у обычных предметов. Значения свойств можно устанавливать и читать.

Например, объект «кнопка» имеет свойства «ширина» и «высота». Значение ширины кнопка запоминает в одном из своих полей. При изменении значения свойства "ширина" вызывается метод, который перерисовывает кнопку.

Кстати, этот пример позволяет сделать важный вывод: свойства имеют первостепенное значение для программиста, использующего объект. Чтобы понять суть и

назначение объекта вы обязательно должны знать его свойства, иногда — методы, очень редко — поля.

Объекты и компоненты

С появлением графических систем программирование пользовательского интерфейса резко усложнилось. Визуальная компоновка и увязка элементов пользовательского интерфейса (кнопок, меток, строк редактора) начали отнимать основную часть времени. Идея визуализировать объекты, объединив программную часть объекта с его видимым представлением на экране в одно целое, стала вполне логичным продолжением. То, что получилось в результате, было названо компонентом.

Компоненты в среде Lazarus — это особые объекты, которые являются строительными кирпичиками визуальной среды разработки и приспособлены к визуальной установке свойств. Чтобы превратить объект в компонент, первый разрабатывается по определенным правилам, а затем помещается в палитру компонентов. Конструируя приложение, вы берете компоненты из Палитры Компонентов, располагаете на форме и устанавливаете их свойства в окне Инспектора Объектов. Внешне все выглядит просто, но чтобы достичь такой простоты, потребовалось создать механизмы, обеспечивающие функционирование объектов-компонентов уже на этапе проектирования приложения! Таким образом, компонентный подход значительно упростил создание приложений с графическим пользовательским интерфейсом.

Классы объектов

Каждый объект всегда принадлежит некоторому классу объектов. *Класс объектов* — это обобщенное (абстрактное) описание множества однотипных объектов. Объекты являются конкретными представителями своего класса, их принято называть *экземплярами класса*. Например, класс СОБАКИ — понятие абстрактное, а экземпляр этого класса МОЙ ПЕС ШАРИК — понятие конкретное.

Для поддержки ООП в язык Lazarus введены *объектные типы* данных, с помощью которых одновременно описываются данные и операции над ними. Объектные типы данных называют *классами*, а их экземпляры — *объектами*.

Классы объектов определяются в секции **type** глобального блока. Описание класса начинается с ключевого слова **class** и заканчивается ключевым словом **end**. По форме объявления классы похожи на обычные записи, но помимо полей данных могут содержать объявления пользовательских процедур и функций. Такие процедуры и функции обобщенно называют *методами*, они предназначены для выполнения над объектами различных операций.

События

Событие — это свойство процедурного типа, предназначенное для создания пользовательской реакции на то или иное воздействие. Если поле процедурного типа, содержит адрес некоторого метода, то присвоить такому свойству значение — значит указать объекту адрес метода, который будет вызываться в момент наступления события. Такие методы называют обработчиками событий.

События имеют разное количество и тип параметров в зависимости от происхождения и предназначения. Все события в Lazarus принято предварять префиксом On: onCreate, onMouseMove, onPaint и т. д. Дважды щелкнув в Инспекторе объектов на странице События в поле любого события, вы получите в программе заготовку метода нужного типа. При этом его имя будет состоять из имени текущего компонента и имени события (без префикса On), а относиться он будет к текущей форме.

Важные события, которые есть почти у каждого визуального компонента:

- **onExit** - возникает, когда компонент теряет фокус ввода;
- **onDblClick** - возникает при двойном щелчке мышкой по компоненту;
- **onKeyDown** - когда при нажатии на кнопку на клавиатуре она оказалась в нижнем положении;
- **onKeyUp** - когда при отпускании клавиатурной кнопки она оказалась в верхнем положении;
- **onKeyPress** - возникает при нажатии на клавиатурную кнопку. От событий **onKeyDown** и **onKeyUp** оно отличается типом используемого параметра **Key** ;
- **onMouseDown** - когда при нажатии кнопки мышки она оказалась в нижнем положении;

- **onMouseUp** - когда при отпускании кнопки мышки она оказалась в верхнем положении;
- **onMouseMove** - возникает при перемещении указателя мышки над компонентом.

Основы ООП

ООП держится на трех китах: инкапсуляции, наследовании и полиморфизме. Для начала о них надо иметь только самое общее представление.

Инкапсуляция

Объединение данных и операций в одну сущность — объект — тесно связано с понятием инкапсуляции, которое означает объединение свойств и методов в объекте. Инкапсуляция делает объекты похожими на маленькие программные модули, в которых скрыты внутренние данные и у которых имеется интерфейс использования в виде подпрограмм.

Наследование

Если вы хотите создать новый класс объектов, который расширяет возможности уже существующего класса, то нет необходимости в переписывании заново всех полей, методов и свойств. Вы объявляете, что новый класс является *потомком* (или *дочерним классом*) имеющегося класса объектов, называемого *предком* (или *родительским классом*), и добавляете к нему новые поля, методы и свойства. Процесс порождения новых классов на основе других классов называется *наследованием*. Новые классы объектов имеют как унаследованные признаки, так и, возможно, новые.

Полиморфизм

Правила контроля соответствия типов языка Object Pascal гласят, что *объекту как указателю на экземпляр объектного типа может быть присвоен адрес любого экземпляра любого из дочерних типов.* Этот принцип называется *полиморфизмом*.

Он означает, что в производных классах вы можете изменять работу уже существующих в базовом классе методов. При этом весь программный код, управляющий объектами родительского класса, пригоден для управления объектами дочернего класса без всякой модификации.

Области видимости

В модели объектов языка Lazarus существует механизм доступа к составным частям объекта, определяющий области, где ими можно пользоваться (области видимости). Поля и методы могут относиться к четырем группам (секциям), отличающимся областями видимости. Методы и свойства могут быть общими (секция public), личными (секция private), защищёнными (секция protected) и опубликованными (секция published). Области видимости, определяемые первыми тремя директивами, таковы.

Поля, свойства и методы секции public не имеют ограничений на видимость. Они доступны из других функций и методов объектов, как в данном модуле, так и во всех прочих, ссылающихся на него.

Поля, свойства и методы, находящиеся в секции private, доступны только в методах класса и в функциях, содержащихся в том же модуле, что и описываемый класс. Свойства и методы из секции private можно изменять, и это не будет сказываться на программах, работающих с объектами этого класса. Единственный способ для кого-то другого обратиться к ним — переписать заново созданный вами модуль (если, конечно, доступны исходные тексты).

Поля, свойства и методы секции protected также доступны только внутри модуля с описываемым классом. Но — и это главное — они доступны в классах, являющихся потомками данного класса, в том числе и в других модулях. Такие элементы особенно необходимы для разработчиков новых компонентов — потомков уже существующих. Оставляя свободу модернизации класса, они все же скрывают детали реализации от того, кто только пользуется объектами этого класса.

Графические команды

Многие компоненты (Form, Image) в Lazarus имеют свойство **Canvas** (канва, холст), представляющие собой область компонента, на которой можно рисовать или отображать готовые изображения.

Отображение графической информации – это часто употребляемый элемент при создании программы.

Каждая точка канвы имеет координаты **X** и **Y** Система координат канвы, как и везде в Lazarus имеет началом левый верхний угол канвы. Координата X возрастает при перемещении вправо, а Y – сверху вниз. Координаты измеряются в пикселях. Пиксель – это наименьший элемент поверхности рисунка.

Рисовать на канве можно следующими способами.

- рисование по пикселям
- рисование с помощью пера Pen.

Рисование по пикселям

Для этого используется свойство канвы *Canvas.Pixels*.

Pixels – это свойство, представляющее собой двумерный массив.

Например: *Canvas.Pixels [10,20]:=clBlack* – это задание пикселю с координатами (10, 20) черного цвета.

Рисование с помощью пера Pen

У канвы имеется *свойство* – перо: *Canvas.Pen*

Этот объект имеет свой ряд свойств:

Pen.Color – определяет цвет пера.

Pen.Style – определяет вид линии:

psSolid - сплошная,	psDachDot - штрихпунктирная,
psDach – штриховая,	psDachDotDot – «тире – две точки»
psDot -пунктирная,	psClear – отсутствие линии.

Pen.Width – ширина линии только для сплошной (по умолчанию = 1).

Pen.Pos – определяет в координатах канвы текущую позицию пера.

Методы:

Canvas.MoveTo (X,Y) - перемещение пера без прорисовки**,** где X и Y – координаты точки, в которую перемещается перо. Эта точка становится исходной.

Canvas.LineTo (X,Y) - провести линию от исходной точки с координатами (X,Y).

Canvas.Rectangle (X1,Y1,X2,Y2 :integer) - точки (X1,Y1) и (X2,Y2) определяют диагональ прямоугольника.

Canvas.Ellipse (X1,Y1,X2,Y2 :integer) – точки (X1,Y1) и (X2,Y2) определяют прямоугольник, описывающий эллипс.

Canvas.Chord (X1,Y1,X2,Y2, X3,Y3,X4,Y4 :integer) – рисует дугу окружности или эллипса и ограниченную хордой, где точки (X1,Y1) и (X2,Y2) – определяют прямоугольник, описывающий эллипс. Начало дуги определяется пересечением прямой проходящей через центр прямоугольника и точкой (X3,Y3). Конец дуги - пересечением прямой проходящей через центр прямоугольника и точкой (X4,Y4).

Canvas.TextOut (X,Y, 'текст') – вывод текста начиная с указанной позиции.

Закрашивание с помощью кисти Bruch

У канвы имеется свойство – кисть *Canvas.Bruch* для закрашивания внутренних областей эллипсов и прямоугольников.

Этот объект имеет свой ряд свойств:

Bruch.Color – определяет цвет, которым заполняется внутренняя область фигуры.

Bruch.Style – определяет стиль заполнения и может принимать значения:

bsSolid – сплошное заполнение цветом		bsDiagCross	
bsFDiagonal		bsHorizontal	
bsBDiagonal		bsVertical	
bsCross		bsClear –без заполнения	

© Составление: Выграненко М.В., vygranenko@mail.ru Апрель 2012

Строковые операции

Символы и строки

Компьютер может обрабатывать не только числовую информацию, но и символьную. Язык Lazarus оперирует с символьной информацией, которая может быть представлена как отдельными символами, так и строками (последовательностью символов).

Символы

Для хранения и обработки символов используются переменные типа Char (в кодировке UTF-8, в которой каждый символ кодируется одним байтом). В этой кодировке символы латинского алфавита, знаки препинания и управляющие символы ASCII записываются кодами US-ASCII, а все остальные символы кодируются при помощи нескольких байтов со старшим битом 1. Из-за этого функции обработки латинских текстов работают как обычно, а с русским текстом при обработке есть проблемы.

Значением переменной символьного типа вообще может быть любой отображаемый символ.

Переменная символьного типа должна быть объявлена в разделе объявления переменных. Инструкция объявления символьной переменной в общем виде выглядит так:

Имя: *char*;

где:

- *имя* - имя переменной символьного типа;
- char - ключевое слово обозначения символьного типа.

Как и любая переменная программы, переменная типа char может получить значение в результате выполнения инструкции присваивания. В этом случае справа от знака := должно стоять выражение типа char, например, переменная типа char или символьная константа - символ, заключенный в кавычки.

Переменную типа char можно сравнить с другой переменной типа char или с символьной константой. Сравнение основано на том, что каждому символу поставлено в соответствие число, можно записать:

'0'<'1'<..<'9'<..<'A'<'B'<..<'Z'<'a'<'b'<..<'z'

В программах обработки символьной информации часто используют функции Chr и Ord. Значением функции Chr является символ, код которого указан в качестве параметра. Функция ord позволяет определить код символа, который передается ей в качестве параметра.

Строки

Строки могут быть представлены следующими типами: shortstring, Longstring и widestring. Различаются эти типы предельно допустимой длиной строки, способом выделения памяти для переменных и методом кодировки символов.

Переменной типа shortstring память выделяется статически, т. е. до начала выполнения программы, и количество символов такой строки не может превышать 255. Переменным типа Longstring и widestring память выделяется динамически - во время работы программы, поэтому длина таких строк практически не ограничена.

Помимо перечисленных выше типов можно применять универсальный строковый тип String. Тип String эквивалентен типу Shortstring.

Переменная строкового типа должна быть объявлена в разделе объявления переменных. Инструкция объявления в общем виде выглядит так:

Имя: **String**;

или

Имя: **String** *[длина]*

где:

- *имя* - имя переменной;
- string - ключевое слово обозначения строкового типа;
- длина - константа целого типа, которая задает максимально допустимую длину строки.

Если в объявлении строковой переменной длина строки не указана, то ее длина задается равной 255 символам.

В тексте программы последовательность символов, являющаяся строкой (строковой константой), заключается в одинарные кавычки.

Используя операции =, <, >, <=, >=, <> переменную типа string можно сравнить с другой переменной типа string или со строковой константой. Строки сравниваются посимвольно, начиная с первого символа. Если все символы сравниваемых строк одинаковые, то такие строки считаются равными. Если в одинаковых позициях строк находятся разные символы, большей считается та строка, у которой в этой позиции находится символ с большим кодом.

Кроме операции сравнения, к строковым переменным и константам можно применить операцию сложения, в результате выполнения которой получается новая строка. Например, в результате выполнения инструкций

first_name: ='Rob';
last_name:='Pike';
ful_name:=first_name+last_name;
переменная ful_name получит значение 'Rob Pike'.

Операции со строками

Так как обработка строк выполняется практически в каждой серьезной программе, стандартно подключаемый модуль System имеет набор процедур и функций, значительно облегчающих этот процесс. Все следующие процедуры и функции применимы и к коротким, и к длинным строкам.

- **Concat**(S1, S2, ... , Sn): string - возвращает строку, полученную в результате сцепления строк S1, S2, ..., Sn. По своей работе функция Concat аналогична операции сцепления (+).
- **Copy**(S: string, Index, Count: Integer): string - выделяет из строки S подстроку длиной Count символов, начиная с позиции Index.
- **Delete**(var S: string, Index, Count: Integer) - удаляет Count символов из строки S, начиная с позиции Index.
- **Insert**(Source: string; var S: string, Index: Integer) - вставляет строку Source в строку S, начиная с позиции Index.
- **Length**(S: string): Integer - возвращает реальную длину строки S в символах.
- **SetLength**(var S: string; NewLength: Integer) - устанавливает для строки S новую длину NewLength.
- **Pos**(Substr, S: string): Byte - обнаруживает первое появление подстроки Substr в строке S. Возвращает номер той позиции, где находится первый символ подстроки Substr. Если в S подстроки Substr не найдено, результат равен 0.
- **Str**(X [: Width [: Decimals]], var S: string) - преобразует числовое значение величины X в строку S. Необязательные параметры Width и Decimals являются целочисленными выражениями. Значение Width задает ширину поля результирующей строки. Значение Decimals используется с вещественными числами и задает количество символов в дробной части.
- **Val**(S: string, var V; var Code: Integer) - преобразует строку S в величину целого или вещественного типа и помещает результат в переменную V. Если во время операции преобразования ошибки не обнаружено, значение переменной Code равно нулю; если ошибка обнаружена (строка содержит недопустимые символы), Code содержит номер позиции первого ошибочного символа, а значение V не определено.

Функция вывода сообщений и диалога с пользователем

Мы уже знаем функцию вывода сообщений showmessage(). Её недостаток в том, что менять в таком окне мы можем только надпись (нельзя менять заголовок окошка, нельзя получить иконку в окошке, нельзя отображать другие кнопки). В Lazarus есть встроенная функция отображения окна сообщения с настраиваемым интерфейсом. Выглядит она так:

MessageDLG(текст_сообщения, тип_сообщения, кнопки, индекс_помощи);

К нашим программам мы пока не пишем дополнительно файлов справки, поэтому **Индекс_помощи** у нас всегда будет нулевым.

Текст_сообщения - строковая величина. Сообщение показывается внутри окна, так же, как при использовании showmessage().

Тип_сообщения - может принимать несколько значений. От этих значений зависит содержимое заголовка и иконка в левом верхнем углу окна. Вид окна для разных типов сообщений можно посмотреть в таблице:

Тип сообщения	Описание	Вид окна
mtWarning	Предупреждающее сообщение. Например, "Вы действительно желаете удалить все данные с диска C:"	
mtError	Обычное окошко вывода сообщения об ошибке.	
mtInformation	Какая-нибудь информация, замечание.	
mtConfirmation	Запрос. Например, на сохранение перед выходом.	
mtCustom	Это сообщение полностью аналогично ShowMessage	

Кнопки - содержит в себе массив кнопок, которые следует показывать в сообщении. Перечень кнопок:

- mbYes
- mbNo

- mbOK
- mbCancel
- mbHelp
- mbAbort
- mbRetry
- mbIgnore
- mbAll

Массив кнопок задается в квадратных скобках []. Например, нам надо задать три кнопки Yes, No, Cancel. Это делается так [mbYes, mbNo, mbCancel]. Поскольку кнопки в сообщении могут быть разные, то MessageDLG является функцией. Она возвращает результат нажатой кнопки.

Соответственно указанным выше кнопкам результат может принимать следующие значения:

- mrNone - окно сообщения закрыто не с помощью кнопки (Alt+F4 или кнопкой "закрыть")
- mrAbort
- mrYes
- mrOk
- mrRetry
- mrNo
- mrCancel
- mrIgnore
- mrAll

Рассмотрим пример. Нам надо спросить у пользователя о дальнейших действиях перед выходом из программы.

1. Сохранить файл.
2. Не сохранять файл.
3. Продолжить редактирование.

```
Var R: Word; // переменная, в которой хранится результат диалога
...
R:=MessageDLG('Сохранить файл?', mtConfirmation, [mbYes, mbNo, mbCancel], 0);
if  R=mrYes then // если нажата кнопка Yes
  begin
    // сохраняем файл и завершаем программу
  end;
if  R=mrNo then // если нажата кнопка No
  begin
    // завершаем работу программы без сохранения
  end;
if  R=mrCancel then // если нажата кнопка Cancel
  begin
    // продолжаем работу без сохранения
  end;
```

Команда вывода сообщения MessageBox

Рассмотренная нами команда MessageDLG - очень гибкая, и у нее есть много достоинств, но есть один существенный недостаток - англоязычный интерфейс.

Следующая команда использует системные сообщения пользователю вашей операционной системы. Вот эта команда:

MessageBox(Handle,ТЕКСТ_СООБЩЕНИЯ,ЗАГОЛОВОК_ОКНА,ТИП_СООБЩЕНИЯ);

Первый параметр - указатель на владельца окна сообщения. Этот параметр вам пока ничего не говорит, устанавливайте его в Handle (это ссылка на окно, откуда это сообщение вызывается).

ТЕКСТ_СООБЩЕНИЯ и ЗАГОЛОВОК_ОКНА - имеют тип PChar, поэтому, во избежание недоразумений и появления неизвестного рода ошибок, выдаваемых компилятором, меняйте тип String в PChar "на ходу". Например:

MessageBox(Handle,PChar('ТЕКСТ_СООБЩЕНИЯ'),PChar('ЗАГОЛОВОК_ОКНА')

Это был перевод из одного типа строковой величины в другой тип.

Параметр ТИП_СООБЩЕНИЯ включает в себя иконку и кнопки.

Кнопки:

- MB_ABORTRETRYIGNORE - кнопки "Прервать", "Повторить", "Пропустить".
- MB_OK - кнопка "Ok".
- MB_OKCANCEL - кнопки "Ok", "Отмена".
- MB_RETRYCANCEL - кнопки "Повторить" и "Отмена".
- MB_YESNO - две кнопки "Да" и "Нет".
- MB_YESNOCANCEL - кнопки "Да", "Нет", "Отмена".

Для того, чтобы отобразить иконку, нужно указать:

- MB_ICONEXCLAMATION
- MB_ICONWARNING
- MB_ICONINFORMATION
- MB_ICONASTERISK
- MB_ICONQUESTION
- MB_ICONSTOP
- MB_ICONERROR
- MB_ICONHAND

Если у вас в сообщении несколько кнопок, а по умолчанию нужно выбрать определенную, то такая кнопка задается:

MB_DEFBUTTON1

- где последняя цифра указывает номер кнопки, выбранной по умолчанию. (Это свойство может быть полезным, например, чтобы обезопасить данные от случайного уничтожения. "Удалить файл?". Две кнопки - "Да", "Нет". По умолчанию мы программно выбираем вторую кнопку. Если пользователь нечаянно сразу нажал на Enter, не осознавая своего поступка, то ничего страшного не произойдет.)

Чтобы указать параметры иконки, кнопок, кнопки по умолчанию в одном параметре ТИП_СООБЩЕНИЯ используется знак «+».

Например:

MessageBox (Handle, PChar ('Выйти из программы?'), PChar ('Мое сообщение'), MB_ICONINFORMATION+MB_OKCANCEL+MB_DEFBUTTON2);

Итак, MessageBox - команда вывода сообщения пользователю, совместимая со всеми языковыми версиями windows. Контроль нажатия на кнопку в MessageBox мы осуществляем аналогично MessageDLG, только возвращаемая величина может принимать следующие значение (соответственно нажатой кнопке):

- IDABORT
- IDCANCEL
- IDIGNORE
- IDNO
- IDOK
- IDRETRY
- IDYES