Е.А.Попов

Экспресс курс программирования в Lazarus

Шестое издание

2011 — 2013 год

Содержание

Введение

Данная книга представляет собой краткий справочник, содержащий необходимые сведения для того чтобы освоить язык Free Pascal и среду Lazarus. Книга рассчитана на уже имеющих опыт программистов, которым необходимо освоить еще одну среду разработки. По ходу изложения дается краткое объяснение терминов и понятий.

Free Pascal является свободным компилятором, который реализует мощный диалект языка Паскаль. Этот диалект совместим с Turbo Pascal и Object Pascal.

Язык Паскаль придумал швейцарский ученый Никлаус Вирт в 1968 году. Своему названию язык обязан французскому математику девятнадцатого века Блезу Паскалю.

Компилятор Free Pascal лежит в основе среды разработки Lazarus, которая предназначена для создания программ с графическим интерфейсом. Lazarus является аналогом среды DELPHI. Среда Lazarus, так же как и лежащий в ее основе компилятор, является свободной.

Автор этого справочника является независимым разработчиком программного обеспечения с многолетней практикой программирования. Причиной перехода на Lazarus явилось необходимость освоения новой среды разработки. В процессе поиска была ориентация на свободную кросс-платформенную среду разработки с широкими возможностями. Наиболее подходящей средой оказался Lazarus. Данная книга возникла после изучения официальной документации как результат желания помочь другим программистам. Надеюсь, она окажется полезной читателю.

Для читателя желательно наличие навыков работы со средами быстрой разработки. В первую очередь книга адресована тем, кто ранее использовал среду Delphi.

Излагаемый материал охватывает основы языка Free Pascal, объектно-ориентированное программирование и создание программ с графическим интерфейсом. Основам языка посвящены первые две части книги. Их изучение необходимо для дальнейшего усвоения материала. В третьей части изложены принципы объектно-ориентированного программирования и освещены реализующие их синтаксические конструкции. Четвертая часть является заключительной. В ней рассмотрено создание программ с графическим интерфейсом.

Часть 1. Основные сведения о языке Free Pascal

Глава 1. Хранение данных

Для чего нужны переменные?

Программы обрабатывают данные. Данные хранятся в переменных. К переменной обращаются при помощи имени. Переменные могут хранить различные данные. Тип переменной определяет хранимые данные. Размер переменной зависит от ее типа и от платформы, на которой выполняется программа.

Объявления переменных

Переменная должна быть объявлена перед использованием.
Синтаксис: var имя:тип;
Замените имя списком имен, чтобы объявить несколько однотипных переменных в одной строке. Имена в списке разделяются запятой.

Константы

Константы отличаются от переменных тем, что не могут изменять значения.
Константа объявляется при помощи следующей конструкции:
const имя=значение;

Целые типы

Тип	Описание	Диапазон	Размер в байтах
Byte	Байт	От 0 до 255	1
shortint	Целое число со знаком	От -128 до 128	1
Smallint	Целое число со знаком	От -32768 до 32767	2
Word	Целое число без знака	От 0 до 65535	2
Longint	Целое число со знаком	От -2147483648 до 2147483647	4
LongWord	Целое число без знака	От 0 до 4294967295	4
Int64	Целое число со знаком	От -9223372036854775808 до 9223372036854775807	8
QWord	Целое число без знака	От 0 до 18446744073709551615	8
Integer	Целое число со знаком	Зависит от платформы	2 или 4
Cardinal	Целое число без знака	От 0 до 4294967295	4

Логические типы

Тип	Размер в байтах	Критерий истинности
Boolean	1	1
ByteBool	1	Ненулевое значение
WordBool	2	Ненулевое значение
LongBool	4	Ненулевое значение

Вещественные типы

Тип	Описание	Диапазон	Размер в байтах
Real	Действительное число	Зависит от платформы	2 или 4
Singe	Число с плавающей точкой	От 1.5E-45 до 3.4E38	2
Double	Число с плавающей точкой	От 5.0E-324 .. 1.7E308	4
Extended	Число с плавающей точкой	От 1.9E-4932 до 1.1E4932	10
Comp	Число с плавающей точкой	От -2E64+1 до 2E63-1	8
Currency	Число с плавающей точкой	От -922337203685477.5808 до 922337203685477.5807	8

Присваивание переменным значения

Используйте оператор присваивания, чтобы присвоить переменной значение.
Синтаксис оператора присваивания: переменная:=значение;
В качестве значения может выступать некоторое число, другая переменная или вызов функции.

Переполнение

Максимальное значение переменной зависит от количества байт, которые выделены для нее. Переполнение возникает при попытке присвоить переменной значение больше максимального. В этом случае в переменную будет записано искаженное значение. Искаженное значение будет меньше того значения, которое вы пытались присвоить переменной.

Логические значения

Вы можете пользоваться идентификаторами True и False при использовании логических переменных. Идентификатор True соответствует истине. Лжи соответствует идентификатор False.

Списки констант

Множество констант удобно объединить в список, называемый перечислением.
Перечисление объявляется при помощи следующей конструкции:
Type имя={константа1:=значение1,...,константаN:=значениеN};

Псевдонимы типов

Вы можете определить псевдоним для уже существующего типа данных. Псевдоним может использоваться при объявлении переменных.

Синтаксис: Type псевдоним=тип;

Ограничение диапазона

Вы можете ограничить диапазон значений уже существующего типа данных.

Синтаксис: Type тип=минимальное значение..максимальное значение;

Явное преобразование типов

Преобразование между встроенными типами данных выполняется автоматически. Выполнить явное преобразование можно, если написать нужный тип и заключить его в круглые скобки. Эта конструкция должна идти перед именем нужной переменной.

Глава 2. Основные операторы

Операторы и операнды

Решаемая программой задача реализуется как набор действий. Действие называют оператором. Для выполнения работы ему необходимы операнды. Унарному оператору нужен один операнд. Бинарные операторы требуют двух операндов. Если для примера взять математическое выражение 5+8, то цифры являются операндами.

Выражения

Выражением называется последовательность операндов и операторов, которая возвращает некоторое значение. Каждое выражение должно оканчиваться точкой с запятой. Вы можете использовать скобки для определения порядка действий в выражениях. Выражения являются основой для вычислений в программах.

Блочный оператор

Оператор, объединяющий в себе другие, называется блочным.

Синтаксис:
begin
операторы
end;

Оператор goto

Оператор goto выполняет переход к указанной метке. Она должна быть объявлена перед определением. Объявление метки располагается там же где и объявления переменных. Определение метки должно находиться в теле подпрограммы или программы.

Синтаксис оператора goto: goto метка;
Синтаксис определения метки: имя:оператор;
Синтаксис объявления метки: label имя;

Математические операторы

Оператор	Описание
+	Сложение
*	Умножение
-	Вычитание
/	Деление
:=	Присваивание
div	Целочисленное деление
mod	Остаток

Операторы сравнения

Оператор	Описание
>	Больше
<	Меньше
<>	Не равно
>=	Больше или равно
<=	Меньше или равно
=	Равно

Логические операторы

Оператор	Описание
not	Логическое Не
and	Логическое И
or	Логическое Или

Поразрядные логические операторы

Оператор	Описание
Xor	Исключающие Или
Shl	Побитовый сдвиг влево
Shr	Побитовый сдвиг вправо
>>	Побитовый сдвиг влево
<<	Побитовый сдвиг вправо

Особенности математических операторов

Оператор деления / применим исключительно для вещественных переменных. Используйте оператор div для выполнения деления целых чисел. Оператор mod применим исключительно для целых переменных.

Глава 3. Условные операторы

Линейные и нелинейные алгоритмы

Алгоритмом называют последовательность действий, приводящая к заданному результату. Алгоритмы бывают линейные и нелинейные.

Линейные алгоритмы представляют собой совокупность однократно выполняемых операций. Их область применения ограничена простыми задачами.

Нелинейные алгоритмы делятся на циклические и разветвляющиеся алгоритмы. Они используются при написании сложных программ.

Разветвляющиеся алгоритмы реализуются при помощи условных операторов.

Принятие решений в программе

Часто бывает необходимо в зависимости от значения переменных выполнять тот или иной код. Для этой цели используются условные операторы.

Условный оператор if

Оператор if выполняет оператор, если условие истинно.
Синтаксис: if условие then оператор

Условный оператор if then else

Синтаксис: if условие then оператор1 else оператор2

Если условие истинно выполняется оператор1. Иначе выполняется оператор2.

Особенности условного оператора if then else

Если оператор if then else содержит два блочных оператора, то в блочном операторе, идущем после ключевого слова end должна отсутствовать точка с запятой. Блочный оператор, идущий после ключевого слова then, имеет стандартный вид.

Оператор case

Оператор case позволяет выполнить разные действия в зависимости от значения переменной. Если значение переменной не совпадает ни с одним из перечисленных, то выполняется оператор, следующий за ключевым словом else. Ключевое слово else и следующий за ним оператор не являются обязательными.

Синтаксис:
case имя переменной of
значение1:оператор1;
…
значениеN:операторN;
else
оператор;
end;

Глава 4. Циклы

Что такое цикл?

Циклы позволяют выполнять операторы несколько раз. Примером необходимости использования цикла является вычисление факториала.

Цикл for

Цикл for выполняет оператор заданное число раз. Оператор for также называют циклом со счетчиком. Синтаксис этого оператора имеет две формы.

Первая форма: for переменная:=начало to конец downto шаг do оператор

Вначале определяется начальное и конечное значение счетчика. Значение счетчика уменьшается на указанное число шагов в конце каждой итерации цикла.

Вторая форма: for переменная:=начало to конец do оператор

Во второй форме не нужно явно указывать шаг. Значение счетчика увеличивается на единицу в конце каждой итерации цикла.

Цикл for..in..do

Оператор for..in..do выполняет код, пока переменная принадлежит множеству. Синтаксис: for переменная in множество do оператор

Цикл repeat..until

Цикл repeat...until выполняет оператор пока условие истинно. Условие проверяется после выполнение оператора.

Синтаксис: repeat оператор; until условие

Если необходимо выполнить несколько операторов, то цикл принимает следующий вид:

repeat
оператор1;
…
операторN
until условие

Как видите в цикле repeat..until можно выполнить несколько операторов не пользуясь блочным оператором.

Цикл while..do

Цикл while..do отличается от repeat..until тем, что проверка условия происходит перед выполнением оператора.
Синтаксис: while условие do оператор

Прерывание цикла и форсирование перехода на следующую итерацию

Для прерывания цикла используйте оператор break. Для перехода на следующую итерацию цикла используйте оператор continue.

Синтаксис оператора break: break;
Синтаксис оператора continue: continue;

Глава 5. Подпрограммы

Виды подпрограмм

Подпрограммой называется некоторый фрагмент программы, который выполняет определенную задачу. Использование подпрограмм позволяет разбить программу на последовательность задач и повторно использовать части кода.

Тело подпрограммы оформляется в виде блочного оператора. Ему может предшествовать объявление необходимых переменных и констант.

Каждая подпрограмма имеет имя, при помощи которого ее можно вызвать. Подпрограммам также можно передавать аргументы.

Подпрограммы делятся на два вида: процедуры и функции. Функция отличается от процедуры тем, что после окончания своей работы возвращает значение.

Вызов подпрограмм

Для вызова подпрограммы используйте следующую конструкцию:

имя подпрограммы(Список аргументов);

Аргументы передаются подпрограмме как значения соответствующих локальных переменных. Аргументы в списке отделяются друг от друга запятой. Оставьте список пустым, если подпрограмме не нужно передавать аргументы.

Определение процедуры

Напишите определение процедуры, чтобы создать ее. Процедура определяется следующим образом:

```
procedure имя процедуры(параметр1;...параметрN);
Объявления переменных и констант
begin
тело процедуры
end;
```

Определение функции

Функция определяется следующим образом:

```
function имя функции(параметр1;...параметрN):тип возвращаемого значения;
Объявления переменных и констант
begin
тело функции
end;
```

Напишите следующую конструкцию в конце тела функции, чтобы возвратить значение: имя функции:=значение.

Имя функции может быть заменено ключевым словом Result.

Так же можно использовать функцию Exit, которая берет в качестве аргумента некоторое значение. Вызов функции Exit приводит к немедленному выходу из вашей подпрограммы и возврату нужного значения.

Вызов функции Exit может располагаться в любом месте тела вашей подпрограммы.

Особенности списка параметров в определениях подпрограмм

Параметры в списке отделяются друг от друга точкой с запятой.
Оставьте список пустым, если подпрограмма не требует передачи аргументов.

Параметр является локальной переменной, то есть доступен только внутри подпрограммы. Переданные при вызове подпрограммы аргументы присваиваются соответствующим локальным переменным.

Параметры-значения

Параметры-значения используются, когда подпрограмма должна работать с копией значений, которые хранятся в аргументах.

Формат объявления параметра: имя:тип

Параметры по умолчанию

Передача аргументов в подпрограмму не является обязательной, если заданы значения по умолчанию.

Формат объявления параметра: параметр:тип=значение

Параметры-переменные

Параметры-переменные применяются, когда подпрограмме необходимо работать напрямую с аргументами и при необходимости изменять их значения.

Формат объявления параметра: var имя:тип

Выходные параметры

Выходные параметры игнорируют начальное значение переданного аргумента и предназначены исключительно для модификации аргументов.

Формат объявления параметра: out имя:тип

Постоянные параметры

Постоянные параметры необходимы, если передаваемые в подпрограмму аргументы не должны изменяться.

Формат объявления параметра: const параметр:тип

Не типизированные параметры

Не типизированные параметры позволяют передавать в подпрограмму аргументы любого типа. При использовании не типизированных параметров не нужно указывать их тип. В качестве не типизированных параметров могут выступать параметры-переменные, постоянные и выходные параметры.

Перезагрузка подпрограмм

Одно имя может соответствовать нескольким подпрограммам. Это называется перезагрузкой. Перезагрузка является практической реализацией полиморфизма. Полиморфизмом называют возможность существования подпрограмм с одинаковым именем, но разным предназначением. Нужно иметь различия в типе или количестве параметров, чтобы использовать перегрузку.

Ассемблерные вставки

Ассемблером называется язык программирования низкого уровня. Каждый диалект ассемблера привязан к определенному процессору или их семейству. Код на языке ассемблера выполняется очень быстро.
Ваша подпрограмма может содержать ассемблерные вставки.

Формат ассемблерной вставки:
Asm
Инструкции
End;

Глава 6. Стандартные математические подпрограммы

Подпрограмма	Описание
Abs(число);	Модуль числа
Arctan(число);	Арктангенс
Cos(число);	Косинус
Sin(число);	Синус
Sqr(число);	Квадрат числа
Sqrt(число);	Квадратный корень числа
Round(число);	Округление числа с плавающей точкой до ближайшего целого
Randomize;	Инициализация генератора случайных чисел
Random(число);	Возвращает случайное значение в диапазоне от 0 до число-1

Глава 7. Инкремент и декремент

Инкрементом называется увеличение значения на единицу.
Декремент выполняет обратное действие. Для этих операций предусмотрены две подпрограммы. Подпрограмма Inc выполняет инкремент. Подпрограмма Dec выполняет декремент. Они берут имя переменной как аргумент.

Глава 8. Работа с консолью

Что такое консоль?

Программы без графического интерфейса работают с консолью. Этим термином называют клавиатуру при выполнении операций ввода данных. Консолью называют дисплей при выполнении операций вывода данных.

Ввод данных с консоли

Чтобы ввести данные с клавиатуры используйте процедуру read или readln. При вызове этих процедур выполнение программы приостанавливается до тех пор, пока пользователь не введет нужные данные.

Ввод заканчивается нажатием клавиши Enter. Переменным присваиваются введенные значения.

Процедура readln переводит курсор на новую строку после окончания ввода.

Обе процедуры берут имя переменной в качестве аргумента. Оно заменяется списком имен, если необходимо ввести несколько значений. Имена в списке отделяются запятой.

При вводе нескольких значений они разделяются пробелом.

Вывод данных на консоль

Для вывода данных на экран используйте процедуру write или writeln. Процедура writeln переводит курсор на новую строку после окончания вывода. Формат передачи аргументов этим процедурам совпадает с форматом процедуры read.

Глава 9. Массивы

Зачем нам массивы?

Массивом называют последовательность однотипных данных. К отдельному элементу массива обращаются по его номеру. Размерностью называют количество элементов массива.

Многомерным называют массив, состоящий из других массивов. Часто используют двумерные массивы. Они представляют собой матрицы. С матрицами вы можете быть знакомы из курса математики. Математики называют таблицу чисел матрицей.

Объявление массива

Синтаксис объявления массива: var имя: array[размерность] of тип;

Размерность указывает количество элементов в массиве.
Формат задания размерности: номер первого..номер последнего элемента
В многомерном массиве следующая размерность указывается после запятой.

Доступ к отдельному элементу массива

К отдельному элементу массива можно обращаться как к переменной, поместив номер элемента в квадратных скобках после имени массива. Для многомерных массивов нужно указывать и следующую размерность.
Предположим, что у нас имеется двухмерный массив с именем matrix. Тогда доступ к элементу первой строки первого столбца осуществляется через конструкцию matrix[1][1].

Открытые массивы

Для использования массивов произвольной длинны в качестве аргументов подпрограмм применяются специальные параметры, называемые открытыми массивами.

Формат объявления параметра: имя: array of тип

Динамические массивы

Динамическим называют массив, память под который выделяется во время работы программы.

Синтаксис объявления динамического массива: var имя: array of тип;

Память под массив выделяется при помощи подпрограммы SetLength.

Формат вызова подпрограммы SetLength: SetLength(имя,размерность).

Размерность задается, как число определяющее количество элементов. В многомерном массиве следующая размерность указывается после запятой.

Память автоматически освобождается, как только динамический массив перестает использоваться. Используйте следующую конструкцию, чтобы освободить память вручную: имя массива:=nil;

Нумерация в динамических массивах

В динамических массивах нумерация подчиняется жестким правилам.
Первый элемент имеет номер 0.

Номер последнего элемента на единицу меньше размерности массива. Выход за границы массива может повредить данные.

Псевдонимы массивов

Для массивов можно определить псевдонимы.

Формат определения псевдонима для обычного массива:
Type псевдоним=array[размерность] of тип;

Формат определения псевдонима для динамического массива:
Type псевдоним=array of тип;

Глава 10. Указатели

Предназначение указателей

Указатель является переменной, которая хранит адрес участка памяти. К указателям можно применять операцию разыменования, которая предназначена для получения значения находящегося по определенному адресу.

Присваивание указателям значения

Значением указателя является адрес. Используйте унарный оператор взятия адреса @, чтобы получить адрес переменной или другой сущности.

Разыменование указателя

Чтобы выполнить операцию разыменования поставьте знак ^ после имени указателя.

Нулевые указатели

Указатель, не содержащий адреса, называется нулевым. Чтобы обнулить указатель, присвойте ему значение nil. Лучше обнулять указатель перед использованием, чтобы избежать повреждения данных. Так же указатель необходимо обнулить после освобождения динамического блока памяти.

Не типизированный указатель

Не типизированный указатель ссылается на данные, тип которых не может быть определен заранее.
Синтаксис объявления для не типизированного указателя: var имя:pointer;

Типизированный указатель

Типизированный указатель ссылается на данные заданного типа.
Синтаксис объявления типизированного указателя: var имя:^тип;

Псевдонимы типизированных указателей

Вы можете определить псевдоним для типизированного указателя.
Синтаксис: Type псевдоним=^тип;

Типизированные указатели как аргументы подпрограмм

Используйте псевдоним типизированного указателя, если хотите использовать его как аргумент подпрограммы или в качестве значения возвращаемого функцией.

Глава 11. Динамическое распределение памяти

Выделение и освобождение блоков памяти

Блок памяти выделяется при помощи подпрограммы GetMem, которая имеет две формы. Первая ее форма берет указатель и размер области памяти в качестве аргументов. Вторая форма берет размер области памяти в качестве аргумента и возвращает указатель на выделенную память. При этом никаких дополнительных действий не производиться.

Освобождение выделенного блока производиться подпрограммой FreeMem, которая имеет две формы. Первая ее форма берет указатель и размер области памяти в качестве аргументов. Вторая форма берет указатель на блок памяти в качестве аргумента.

Контроль поведения при динамическом выделении памяти

Как бы много не было памяти, ее все же может не хватить. При нехватке памяти программа может выбрасывать исключение или обнулять указатель.

Исключением называется ошибка, возникшая во время работы программы. Об исключениях и модулях вы узнаете в следующей части этой книги.

Присвойте глобальной переменной ReturnNilIfGrowHeapFails значение True, чтобы обнулять указатель при нехватке памяти. Эта переменная объявлена в модуле System, который подключается автоматически.

Подпрограммы для работы с памятью

Подпрограмма	Описание
MemSize(блок)	Возвращает размер блока памяти
AllocMem(размер)	Выделяет блок памяти и возвращает указатель на него. Блок заполняется нулями
SizeOf(элемент)	Возвращает размер элемента
addr(аргумент)	Возвращает адрес аргумента
New(указатель)	Выделяет память под элемент
Dispose(указатель)	Освобождает выделенную память

Глава 12. Процедурный тип

Указатели на подпрограммы

Прежде чем использовать указатель на подпрограмму нужно определить новый тип данных. Этот тип называется процедурным.

Определение для процедуры:
type имя:=procedure(список параметров процедуры);

Определение для функции:
type имя:=function(список параметров функции):тип возвращаемого значения;

Объявление указателя на подпрограмму выполняется при помощи следующей конструкции: var указатель:процедурный тип;

Присваивание указателям адреса и вызов подпрограмм

Формат присваивания указателю адреса: указатель:=@имя подпрограммы;
Формат вызова подпрограммы через указатель: указатель(аргументы);

Глава 13. Множества

Понятие множества

Понятие множества в Паскале аналогично тому, что принято в математике.

Создание экземпляра множества

Экземпляр множества создается аналогично обычной переменной.

Синтаксис: var имя: set of тип;

Заполнение множеств

Чтобы заполнить множество элементами воспользуйтесь следующей конструкцией: имя множества:=[список];

Элементы списка отделяются друг от друга запятой. Оставьте список пустым если множество не содержит элементов.

Ограничения множеств

Множество может не содержать элементов. В таком случае его называют нулевым. Максимальный размер множества ограничен 255 элементами.

Добавление и удаление элементов

Элемент добавляется во множество при помощи подпрограммы Include.

Синтаксис: Include(множество,элемент);

Для удаления элемента из множества используйте подпрограмму Exclude.

Синтаксис: Exclude(множество,элемент);

Объединение множеств

Операция объединения возвращает новое множество, состоящее из элементов входящих хотя бы в одно множество из пары множеств.

Синтаксис: множество1+множество2;

Пересечение множеств

Операция объединения возвращает новое множество, состоящее из элементов входящих в обе пары множеств.

Синтаксис: множество1*множество2;

Разность множеств

Операция объединения возвращает новое множество, состоящее из элементов первого множества, которые не входят во второе.

Синтаксис: множество1-множество2;

Проверка элемента

Чтобы определить принадлежность элемента к множеству используйте оператор бинарный in. В качестве первого операнда используйте элемент. Вторым операндом идет множество. Оператор in возвращает истинное значение, если элемент принадлежит множеству.

Глава 14. Обработка строк

Хранение строк и символов

Одиночный символ храниться в переменной типа char. Строка представляет собой последовательность символов. Она храниться в переменной типа string.

Склейка строк

Для склейки нескольких строк в одну используйте оператор суммирования +.

Присваивание значений

Значение, которое присваивается строковой переменной, должно быть заключено в одинарные кавычки.

Ограничение длины строк

Вы можете задать длину строковой переменной при ее объявлении.

Синтаксис: var имя:string[длина];

Подпрограммы для работы с символами

Функция chr возвращает символ с заданным кодом, а функция ord выполняет противоположенное действие.

Процедуры преобразования

Для преобразования числа в строку используйте процедуру Str. Она берет в качестве аргументов строку и имя числовой переменной.

Для преобразования строки в число используйте процедуру Val.

Формат вызова процедуры Val: Val(Строка, число, код);
Число и код являются именами переменных. Число содержит результат преобразования. Код будет содержать нулевое значение или позицию строки, в которой произошла ошибка преобразования.

Подпрограммы для работы со строками

Подпрограмма	Описание
Length(строка);	Возвращает длину строки
SetLength(строка,длина);	Изменяет длину строки
LowCase(строка);	Преобразует строку в нижний регистр
UpCase(строка);	Преобразует строку в верхний регистр
Copy(строка,позиция,размер);	Возвращает часть строки, то есть подстроку
Pos(подстрока или символ,строка);	Возвращает позицию, в которой находиться подстрока или символ
Delete(строка, позиция, количество символов)	Удаляет часть строки

Глава 15. Записи

Запись как пользовательский тип данных

Вы можете определить свой тип данных при помощи записей. Запись представляет собой сущность, которая содержит и объединяет в себе разные переменные. Переменные внутри записи называют полями.

Описание записи

Синтаксис описания записи:

Type имя записи=record
Объявления полей
End;

Формат объявления поля: поле:тип;

Объявление экземпляра записи и доступ к отдельным полям

Экземпляр записи объявляется как обычная переменная. Но вместо стандартного типа указывается имя записи.

С полем экземпляра записи можно работать почти также как с обычной переменной. Формат обращения к полю экземпляра записи: экземпляр.поле

Использование записей как аргументов подпрограмм

Запись может быть использована, как аргумент подпрограммы, если в списке параметров использовать имя записи вместо стандартного типа.

Глава 16. Перезагрузка операторов

Необходимость перезагрузки операторов

Записи позволяют программисту определить собственный нестандартный тип данных. При работе с записями бывает необходимо переопределить поведение операторов. Этот процесс называется перезагрузкой операторов.

Реализация перезагрузки

Перезагрузка операторов реализуется при помощи следующей конструкции:

operator оператор(Список параметров) результат: тип результата;
begin
тело оператора
end;

Эта конструкция напоминает определение подпрограммы. Параметры задаются так же как параметры подпрограмм. Результат является локальной переменной. Значение результата будет возвращено оператором после выполнения работы.

Глава 17. Определение типа во время выполнения программы

Иногда бывает необходимо иметь переменную, тип которой определяется во время выполнения программы.

Для таких целей предусмотрен специальный тип данных Varriant. Переменным типа Varriant можно присваивать целочисленные и дробные переменные, а так же строки и интерфейсы.

Глава 18. Работа с файлами

Типы файлов

Файлы бывают следующих типов: текстовые, типизированные и двоичные. Типизированные файлы состоят из записей указанного типа.

Проверка наличия ошибок

Во время работы с файлами могут происходить ошибки. Функция IOResult не берет аргументов и возвращает ноль при отсутствии ошибок в последней выполненной операции с файлом.

Объявление файловой переменной

С каждым файлом должна быть связана файловая переменная.

Объявление переменной для текстового файла: var имя:text;
Объявление переменной для типизированного файла: var имя:file of тип;
Объявление переменной для двоичного файла: var имя:file;

Связывание переменной с файлом

Перед открытием требуется связать файловую переменную с файлом. Для связывания воспользуйтесь процедурой Assign. В качестве аргументов она берет файловую переменную и строку с именем файла.

Открытие и закрытие файла

Прежде чем работать с файлом его нужно открыть. Для открытия файла в режиме чтения используйте процедуру Reset. Для открытия файла для записи используйте процедуру Rewrite. Для того чтобы открыть файл для записи данных в конец файла применяется процедура Append.

Когда работа с файлом закончена, используйте процедуру Close, чтобы закрыть файл. Все процедуры берут в качестве аргументов файловую переменную.

Чтение и запись текстовых данных

Для чтения и записи в текстовые файлы используются подпрограммы Read и Write. При их вызове перед списком аргументов указывается файловая переменная. Файловая переменная и список параметров отделяются друг от друга запятой. Так же можно использовать подпрограммы Readln и Writeln.

Чтение и запись данных в типизированные и двоичные файлы

Для чтения данных из типизированного или двоичного файла используется процедура BlockRead.

Для записи в типизированный или двоичный файл используется процедура BlockWrite.

Формат вызова BlockRead: BlockRead(файл,буфер,количество байт);
Формат вызова BlockWrite: BlockWrite(файл,буфер,количество байт);

Первый аргумент является файловой переменной.

Буфер представляет собой имя переменной, которая будет использована как посредник при операциях чтения или записи.

Подпрограммы для работы с файлами

Подпрограмма	Описание
FileSize(файловая переменная);	Возвращает размер файла
FilePos(файловая переменная);	Возвращает текущую позицию в файле
EOF(файловая переменная);	Определяет, достигнут ли конец файла
Seek(файловая переменная, позиция);	Меняет файловую позицию

Подпрограммы для работы с файловой системой

Подпрограмма	Описание
ChDir(каталог);	Смена текущего каталога
MkDir(каталог);	Создание каталога
RmDir(каталог);	Удаление пустого каталога
Rename(файловая переменная,имя);	Переименовывает файл
Erase(файловая переменная);	Удаляет файл

Глава 19. Структура программы

Программа на паскале начинается с заголовка. Он не является обязательным.

Формат заголовка: program заголовок;

Далее может идти подключение модулей.

Затем следует раздел описаний, в котором находиться объявления глобальных переменных и констант, а так же меток и записей.

Далее идет раздел определений. В нем находятся определения подпрограмм. Затем идет тело программы. Оно состоит из операторов.

Тело программы выглядит следующим образом:
begin
операторы
end.

Глава 20. Области видимости данных

С переменными и константами связано понятие видимости. Оно определяет их доступность в программе. Переменные и константы бывают локальные и глобальные.

Локальные переменные и константы доступны только внутри подпрограммы, в которой они объявлены. Глобальные переменные и константы доступны из любой точки программы. Подобное деление применимо также к массивам.

Часть 2. Сопровождение и повышение надежности программ

Глава 1. Комментарии

Комментарием называется поясняющий текст. Текст комментария игнорируется при компиляции. Комментарии могут состоять из одной или нескольких строк. Комментарий из одной строки начинается с двойной косой черты(//). За ней идет текст комментария.

Комментарий из нескольких строк помещается между открывающей и закрывающей фигурной скобкой.

Глава 2. Завершение программы

Нормальное завершение

Когда программа завершает свою работу она передает операционной системе код возврата. По нему операционная система может узнать результат работы программы. Вы можете сами определить код возврата. Для этого присвойте значение глобальной переменной ExitCode. Используйте целое число в качестве значения. Допустимы отрицательные числа. Переменная ExitCode объявлена в модуле System.

Аварийное завершение

Воспользуйтесь процедурой Halt для того чтобы форсировать завершение работы. В качестве аргумента процедура берет код возврата.

Глава 3. Параметры командной строки

Часто бывает удобно предусмотреть возможность передачи программе параметров при запуске. Параметры передаются в командной строке. Возможность передачи параметров улучшает взаимодействие с пользователем. Кроме того вы обеспечите возможность интеграции вашей программы с другими. Для того чтобы узнать количество переданных параметров используйте функцию ParamCount, которая не берет аргументов. Для того чтобы узнать значение конкретного параметра используйте функцию ParamStr. Она берет в качестве аргумента номер параметра.

Глава 4. Обработка ошибок

Исключения

Во время выполнения программы могут возникать ошибки. Их называют исключениями. Эта глава посвящена перехвату и обработки исключений.

Выброс исключений

Выбросом исключений называется информирование операционной системы о произошедшей ошибке. После этого операционная система выводит информацию об ошибке.

Для выброса исключений используется унарный оператор raise.

Формат оператора raise: raise исключение;

Список исключений можно посмотреть в официальной документации по Free Pascal и Lazarus.

Обработка и перехват исключений

Для обработки исключений существуют две конструкции: try...except и try...finally.

Формат конструкции try...except:

Try
Код, который может вызвать ошибки
Except
On тип исключения do Код, который обрабатывает исключение
end;

Если во время выполнения программы возникают ошибки, то выполнение кода вызвавшего ошибку приостанавливается и выполняется код обрабатывающий исключение.

Формат конструкции try...finally:

Try
Код, который может вызвать ошибки
Finally
Код, который обрабатывает исключение
end;

Здесь вам не нужно указывать тип исключения. Если во время работы исключение не возникает, то вначале выполняется код из секции Try, а затем код из секции Finally. Если исключение произошло, то сразу выполняется код из секции Finally. Код из секции Try игнорируется.

Глава 5. Модули

Разделение программы на модули

Используемые подпрограммы можно вынести в отдельный файл, который называют модулем. Такой подход облегчает модификацию программ и позволяет повторно использовать ранее написанный код.

Стандартный модуль System

Модуль System имеет отличие от остальных модулей. Он автоматически подключается к программе. Таким образом, для его подключения не нужно использовать ключевое слово uses.

Подключение модулей

Модуль необходимо подключить перед использованием.
Формат подключения модуля: uses имя;
Вместо имени модуля можно использовать список модулей. Список состоит из имен разделенных запятой.

Альтернативный вариант подключения модулей

В приведенном выше варианте подключение модулей компилятор самостоятельно ищет модули. Если вы хотите явно указать имя файла с модулем, то воспользуйтесь альтернативным вариантом подключения модулей.

Альтернативный формат подключения модуля: uses имя in 'файл';

Имя файла может быть полным, то есть содержать путь к файлу. В качестве файла укажите файл с исходным кодом модуля.

Написание модулей

Вы можете сами написать модуль. Структура модуля следующая:

unit имя модуля;
interface
Раздел описаний
implementation
Раздел реализации
end.

Раздел описаний содержит объявления необходимых переменных и прототипы подпрограмм. Прототипы идут после объявления переменных. Прототипы подпрограмм отличаются от определений отсутствием тела. В разделе реализации находятся определения подпрограмм.

Зависимости

Если в вашем модуле используется другой модуль, то он подключается в начале раздела описаний. Рекурсивные зависимости недопустимы.

Секции инициализации и завершения

Эти секции не являются обязательными и могут быть пропущены.
Но если вы определили секцию инициализации, то вы обязаны дать определение секции завершения. Секция инициализации не может быть пустой. Она содержит код, выполняемый при первом обращении к модулю. Код из секции завершения выполняется при завершении работы программы. Секция инициализации идет сразу после раздела реализации.

За секцией инициализации идет секция завершения.

Формат описания секции инициализации:

Initialization
Код
end.

Формат описания секции завершения:

Finalization
Код
end.

Часть 3. Объектно-ориентированное программирование

Глава 1. Принципы объектно-ориентированного программирования

Парадигмой называется подход к программированию. Наиболее часто применяют процедурный и объектно-ориентированный подход.

Суть процедурного программирования состоит в разбивке исходной задачи на совокупность связанных подпрограмм. Этот прием называют функциональной декомпозицией. Термин «процедурное программирование» сейчас заменен термином «структурное программирование». Но смена названия не меняет суть. Структурное программирование было рассмотрено в первых двух частях книги.

В объектно-ориентированном программировании данные и подпрограммы их обрабатывающие объединяются в одну сущность называемую объектом. При этом обрабатываемые данные называют членами. Подпрограммы внутри объектов называют методами. Объединение данных и методов внутри объекта называют инкапсуляцией.

Взаимодействие объекта с другими частями программы можно ограничить при помощи спецификаторов доступа.

Порождения одного объекта от другого называют наследованием. Класс является основой для создания объекта.

Глава 2. Классы

Спецификаторы доступа

Спецификаторы доступа позволяют ограничить доступ к элементам класса.
Список доступных спецификаторов дан в таблице ниже.
Если спецификатор доступа не задан, то используется Public.

Спецификатор	Описание
Private	Класс доступен только в модуле, который его описывает
Strict Private	Члены и методы доступны внутри класса
Protected	Элементы класса доступны в производном классе из другого модуля
Public	Ограничений нет.
Published	Аналогично Public, но запрещает массивы в качестве членов класса

Описание класса

Синтаксис описания класса:

Type имя=class
Список элементов класса
end;

Описание класса дается в разделе объявлений перед объявлением используемых в программе переменных и констант.

Список элементов имеет следующий вид:

спецификатор доступа:
Объявления членов и прототипы методов

Вначале объявляются члены. Вслед за ними описываются прототипы методов. Член объявляется как переменная. Прототип метода похож на определение подпрограммы, но у него отсутствует тело.

Определения методов

После описания прототипов методов необходимо дать их определение. Определения методов даются за пределами описания класса в разделе определений перед определением используемых подпрограмм.

Определения методов похожи на определения подпрограмм с небольшим отличием. После ключевого слова function или procedure идет имя класса. А затем после точки идет имя метода.

Доступ к членам и вызов методов

Каждый объект содержит собственную копию членов определенных в классе. Доступ к ним осуществляется так же как к полям экземпляра структуры.

Вызов методов осуществляется следующей конструкцией:

объект.метод(список аргументов);

Конструкторы и деструкторы

В каждом классе должно быть два специальных метода: конструктор и деструктор. Конструктор вызывается при создании объекта. Деструктор вызывается при уничтожении объекта. Имена этих методов произвольны. Конструктор может принимать аргументы на вход. Деструктор их не имеет аргументов.

Отличие описания прототипа и определения конструктора от обычного метода состоит в том, что вместо ключевого слова procedure или function вы используйте ключевое слово Constructor.

Отличие описания прототипа и определения деструктора от обычного метода состоит в том, что вместо ключевого слова procedure или function вы используйте ключевое слово Destructor.

Создание объекта

Перед тем как создать объект его необходимо объявить. Объявляется объект так же как переменная с тем отличием, что вместо типа указывается имя класса. После объявления объекта необходимо вызвать конструктор.

Вызов конструктора и деструктора

Память под объекты, которые являются представителями класса, выделяется во время работы программы. Выделение памяти происходит автоматически при вызове конструктора. Освобождение памяти происходит при вызове деструктора.

Формат вызова конструктора: объект:=класс.конструктор(список аргументов);

Деструктор вызывается как обычный метод.

Проверка объекта на принадлежность к классу

Чтобы проверить объекта на принадлежность к определенному классу используется бинарный оператор is.

В качестве первого аргумента он берет имя объекта, а вторым аргументом является класс. Результатом работы этого оператора является логическое значение. Если объект объявлен, но не создан, то оператор возвращает False. На практике оператор is используется вместе с условным оператором if.

Свойства

Свойства позволяют ограничить доступ к членам. Ограничения задаются при помощи спецификаторов доступа. Спецификатор read дает права чтения. Спецификатор write дает права записи. Если спецификатор не задан, то по умолчанию даются права на чтение.

Объявление свойства осуществляется следующим образом:

property имя:тип ограничитель;

Синтаксис задания ограничителя: спецификатор доступа:идентификатор

Ограничитель может быть обычным и сдвоенный. Сдвоенный ограничитель задается как два ограничителя разделенные пробелом.

Идентификатор представляет собой имя члена, который привязывается к свойству. Идентификатор так же может быть именем метода, который работает с членом. Член или метод должны быть ранее объявлены в классе.

Доступ к члену осуществляется через имя свойство или метод.

Разновидности методов

Существуют несколько разновидностей методов. Вид метода определяется при помощи спецификатора. Вы можете явно задать разновидность метода при определении класса. Разновидность метода задается в его прототипе.

Синтаксис: прототип метода(список параметров); спецификатор;

Список спецификаторов дан в таблице ниже.

Спецификатор	Описание
Virtual	Виртуальный метод.
Abstract	Абстрактные методы не имеют реализации и применяются в родительских классах
Dynamic	Синоним Virtual

Метод является обычным, если явно не указан его спецификатор.

Размещение классов в модулях

Классы можно размещать в модулях. При размещении класс внутри модуля запомните простые правила. Описание класса располагается в разделе описания модуля. Определения методов находятся в разделе реализации модуля.

Глава 3. Наследование

Простое наследование

Класс может порождаться от другого класса, который называется родительским. Как вы помните, этот процесс называется наследованием. Если класс имеет одного родителя, то наследование называют простым. При наследовании конструкция описывающая класс меняется.

Описание класса при наследовании выглядит следующим образом:

Type имя=class(родительский класс);
Список элементов класса
end;

Множественное наследование

Если класс имеет нескольких родителей, то такое наследование называют множественным. При множественном наследовании имя родительского класса заменяется списком родителей, в котором имена классов разделены запятой.

Перезагрузка методов в производном классе

Перезагрузка методов аналогична перезагрузке подпрограмм.

Для того чтобы перезагрузить метод в производном классе вставьте его прототип в объявление производного класса.

Для перезагружаемых методов вы можете использовать спецификатор overload, который введен для совместимости с Delphi. Если производный класс с перезагружаемыми методами находиться в одном модуле с базовым классом, то использование спецификатора overload не обязательно.

Отличия обычных и виртуальных методов

Виртуальные методы часто применяются при перезагрузке методов. В основе работы виртуальных методов лежит механизм позднего связывания. Он прямо противоположен раннему связыванию. При раннем связывании привязка методов к объектам происходит на этапе компиляции программы.

Если применяется динамическое связывание, то привязка методов к объектам осуществляется во время работы программы при вызове конструктора. Виртуальный метод не может быть перекрыт обычным методом. Метод является обычным, если явно не указан спецификатор virtual или dynamic.

Перезагрузка виртуальных методов

Для того чтобы перезагрузить виртуальный метод в производном классе используйте в прототипе метода спецификатор override.

При этом перегруженный метод так же будет виртуальным. Если вы хотите чтобы перегруженный метод являлся обычным методом, то используйте вместо спецификатора override спецификатор reintroduce.

Глава 4. Вспомогательные классы

Расширение функционала

Вспомогательные классы позволяют расширить функционал существующего класса без необходимости создания производного. Вспомогательный класс не является полноценным классом. Он имеет ряд жестких ограничений. Они связаны с особенностями вспомогательных классов. Ограничения вспомогательного класса:

1.Запрещены конструкторы и деструкторы.
2.Невозможно определить члены и свойства
3.Запрещены абстрактные методы.
4.При перезагрузке виртуальных методов используется спецификатор overload.

Создание вспомогательного класса

Синтаксис описания вспомогательного класса:

Type имя=class helper of класс;
прототипы методов;
End;

Определение методов аналогично определению в обычном классе.

Вызов методов

Поскольку назначение вспомогательного класса ограничиваться лишь расширением функциональности, есть одно отличие при вызове методов вспомогательного класса. При вызове методов надо указывать имя объекта являющегося экземпляром того класса, который указан после ключевого слова of.

Глава 5. Ссылки на классы

Что такое ссылка на класс?

В диалекте Free Pascal есть специальный тип данных, называемый ссылкой на класс. Переменным этого типа можно присваивать родительский класс и производные от него классы.

Использование ссылок на классы в Free Pascal

Синтаксис описания ссылки на класс:

Type имя ссылки на класс=class of имя класса;

Объявление переменной делается обычным образом, но вместо встроенного типа указывается ссылка на класс. Этой переменной можно присвоить класс и все производные от него классы.

Глава 6. Интерфейсы

Поддержка интерфейсов

Free Pascal поддерживает интерфейсы. Они необходимы если вы хотите использовать объекты, которые находятся в бинарных файлах. Примером такого файла является динамически подключаемая библиотека.

Интерфейс, так же как и класс описывает объект с тем отличием, что он не содержит реализации методов. Интерфейс содержит только прототипы.

Отличия от классов

Отличия интерфейса от класса состоят в следующем:

1.Интерфейсы можно использовать, только если переключить компилятор в режим совместимости с Delphi или Object Pascal.
2.Нельзя использовать спецификаторы доступа. Все члены и методы интерфейса доступны публично.
3.Конструкторы и деструкторы не используются. Интерфейс не предназначен для непосредственного создания.

4.Спецификаторы, определяющие разновидности методов, так же не используются.

Описание интерфейса

Интерфейс описывается при помощи следующей конструкции:

Type имя=Interface
список элементов
End;

Идентификатор интерфейса

Приведенная ниже информация специфична для Windows и не актуальна для других систем.

В Windows для того чтобы без ограничений пользоваться объектами, которые описывает интерфейс, он должен иметь идентификатор.

Идентификатор представляет собой 128 битное число. Такие идентификаторы обозначают термином GUID.

GUID определяется при помощи следующей конструкции:
const имя:TGUID='значение';

Определение GUID находиться в разделе описаний модуля или программы.

Реализация интерфейсов

Интерфейс реализуется при помощи класса. Его объявление должно идти сразу после описания интерфейса. В определении класса описание элементов должно совпадать с их описанием в интерфейсе.

Для реализации интерфейса нужно два родительских класса.

В качестве первого родителя используйте класс TinterfacedObject. Имя интерфейса используйте как второго родителя.

Глава 7. Объекты без классов

Объекты сами по себе

Free Pascal позволяет определить объекты, у которых нет классов. По сути, объект без класса является модификацией записи. Существует два больших различия между классами и объектами без класса. Память под объект без класса выделяется из стека. Память под класс выделяется из кучи. При использовании классов обязательно вызывать конструктор, который производит

инициализацию объекта принадлежащего классу. Так же не следует забывать вызывать деструктор по окончанию использования класса.

В случае использования объектов без классов применение конструктора и деструктора необходимо только при присутствии в объекте виртуальных методов.

Спецификаторы доступа

Спецификатор	Описание
Private	Объект доступен только в своем модуле
Protected	Объект доступен в своем модуле и в других модулях, которые на него ссылаются
Public	Объект доступен везде

Описание объекта

Синтаксис описания объекта:

Type имя=object
Список элементов
End;

Описание объекта дается в разделе объявлений перед объявлением используемых в программе переменных и констант.

Определение методов

Определения методов в объектах без класса, похоже на определение методов классовых объектов. Но вместо имени класса используется имя объекта. Определения методов даются за пределами описания объекта в разделе определений перед определением используемых подпрограмм.

Размещение объектов в модулях

Правила аналогичны размещению классов в модулях.

Объявление экземпляра объекта и манипуляции с ним

Вначале нужно объявить экземпляр, а затем вызвать конструктор.
Обь явление экземпляра осуществляться при помощи следующей конструкции:
var экземпляр:объект;

Вызов методов и обращение к данным членам делается аналогично объектам использующим классы, но вместо имени объекта используется имя экземпляра. Конструктор и деструктор вызывается как обычный метод.

Статистические поля

Статистические члены похожи на глобальные переменные доступные только внутри объекта. Объявление членов находиться в списке элементов. Чтобы объявить член как статистический используйте спецификатор static.

Оператор with

Обычно когда вы вызываете методы или обращаетесь к членам объекта то используйте конструкцию вида объект.обращение; для объектов являющимися представителями класса или экземпляр.обращение; для объектов без классов.

Оператор with позволяет более удобно манипулировать объектами.
Синтаксис:
with объект do
begin
код, манипулирующий объектом
end;

В качестве объекта может выступать представитель класса или обычный объект. Так же оператор with позволяет манипулировать экземпляром записи.

Полиморфизм в объектах

Объекты без классов также поддерживают наследование. Родительские объекты задаются при определении объекта. Задание родительских объектов аналогично заданию родительских классов. Соответственно в объектах доступна перезагрузка и использование виртуальных методов. Для объектов без классов в прототипе перезагружаемых методов указывается спецификатор virtual.

Вызов методов производного объекта в родительском объекте

Для того чтобы вызвать метод производного объекта в экземпляре родительского необходимо использовать оператор as. Он приводит тип экземпляра родительского объекта к типу производного.

Синтаксис: объект1 as объект2.

Если объект2 не является производным от объект1, то выбрасывается исключение. Практически оператор as применяется в связке с оператором with. Оператор as так же применим к объекту, который является представителем класса.

Глава 8. Динамические объекты

Особенности динамических объектов

По умолчанию память под объекты без класса не распределяется динамически. Она выделяется при запуске программы и освобождается перед ее завершением. Это влечет за собой большее потребление памяти. Часто потребление памяти является важным фактором. Поэтому Free Pascal предоставляет возможность ручного создания и работы с динамическими объектами. Для динамических объектов необходимо наличие конструктора и деструктора.

Опытный программист заметит схожесть механизма работы с динамическими объектами между диалектами Free Pascal и Turbo Pascal.

Создание объектов

Перед созданием объекта нужно объявить указатель на объект. Он объявляется так же как обычный, но вместо встроенного типа используется имя объекта. Создание объекта происходит при выделении памяти для него. Память выделяется при помощи подпрограммы New.

Делается это двумя способами. Рассмотрим их оба.

Первый способ заключается в передачи подпрограмме New указателя в качестве аргумента. Затем нужно вызвать конструктор.
Формат вызова конструктора: указатель^.конструктор(параметры);

При использовании второго способа конструктор вызывается автоматически. При этом меняется формат вызова подпрограммы New.

Первая форма вызова подпрограммы New: New(указатель,конструктор);
Вторая форма вызова подпрограммы New: указатель:=New(объект,конструктор);

Удаление объектов

Объект удаляется при помощи подпрограммы Dispose.
Формат вызова: Dispose(указатель, деструктор);
Подпрограмма Dispose автоматически вызывает деструктор.

Работа с объектами

Синтаксис обращения к членам и вызов методов немного меняется, если объект является динамическим.

Обращение к члену объекта: указатель^.член;

Вызов метода: указатель^.метод(аргументы);

Особенности создания и удаления объектов

Конструктор не должен иметь список параметров при использовании второго способа вызова подпрограммы New. Кроме того в качестве второго аргумента передается только имя конструктора. Аналогичные правила действуют при передаче деструктора в подпрограмму Dispose.

Поясним все вышеизложенное на примере.

Предположим, существует объект test и указатель на него point. Объект test имеет конструктор setup и деструктор kill.

Примеры правильного создания объекта:
New(point,setup); и point:=New(test,setup);

Примеры неправильного создания объекта:
New(point,point^.setup); и point:=New(test,point^.setup);

Пример правильного удаления объекта:
Dispose(point,kill);

Пример неправильного удаления объекта:
Dispose(point,point^.kill);

Часть 4. Создание программ с графическим интерфейсом

Глава 1. Средства быстрой разработки

В настоящие время широко используются программы с графическим интерфейсом. Для повышения скорости создания таких программ применяют среды быстрой разработки. В англоязычной литературе для обозначения этих сред применяют термин RAD.

Среды быстрой разработки являются развитием концепции интегрированных сред разработки.
Интегрированная среда разработки представляет собой пакет программ. В нем помимо компилятора и компоновщика, содержаться редактор исходного кода и отладчик. Редактор исходного кода представляет собой текстовый редактор с подсветкой синтаксиса.

В основе быстрых средств разработки лежит идея визуального проектирования интерфейса при помощи редактора форм. Сама работа программы определяется совокупностью обработчиков событий.

Событием называется действие пользователя или операционной системы.

В средах быстрой разработки каждый элемент интерфейса представляет собой объект класса. Обработчик события представляет собой метод этого класса.

Среда Lazarus совмещает обе концепции.

В Lazarus вы можете создавать консольные и графические программы.

К практическим достоинствам Lazarus относиться поддержка нескольких платформ. Это выражается в том, что Lazarus и программы, созданные с его использованием, работают на многих системах. Разработчики постоянно работают над поддержкой новых платформ. Библиотека Lazarus Component Library используемая в среде Lazarus для создания программ с графическим интерфейсом. Она так же поддерживает несколько платформ.

Глава 2. Описание элементов графического интерфейса

Графический интерфейс

Большинство современных операционных систем позволяют пользователям использовать два типа интерфейсов: консольный и графический.

В консольном интерфейсе работа с компьютером осуществляется при помощи ввода текстовых команд. Первые операционные системы были рассчитаны исключительно на консольный интерфейс.

В графическом интерфейсе программы и данные представлены в виде графических элементов. Идея графического интерфейса возникла в корпорации Xerox. Идея возникла в 1970 году и была воплощена в компьютере Xerox Alto, который не получил широкого распространения. Позже свою реализацию графического интерфейса предложили фирмы Apple и Microsoft. После этого графический интерфейс получил широкое распространение и стал реализовываться во многих системах.

Первоначально графический интерфейс был ориентирован на мышь, но сейчас он успешно применяется в устройствах с сенсорными экранами.

Стандартизация

Несмотря на все разнообразие графических интерфейсов, они имеют общие элементы и схожим образом выглядят в разных системах. При помощи этих элементов пользователи взаимодействуют с программами. Таким образом, облегчается и ускоряется освоение программ. Давайте рассмотрим основные элементы графического интерфейса.

Окно

Окно является основным элементом программы. В нем расположены другие элементы. Окно имеет заголовок, в котором обычно написано название программы или имя открытого файла.

Поле ввода

Это поле нужно для ввода информации и часто выглядит как белый квадрат.

Флажок

Флажок предназначен для выбора нескольких пунктов. Он представляет собой квадрат с текстом напротив. Флажок активирован если в квадрате стоит галочка.

Полоса статуса

Полоса статуса представляет собой полосу с текстом, которая расположена внизу окна. Она имеет ширину равную ширине окна и предназначена для информирования пользователя.

Меню

Меню расположено сразу под заголовком программы. Меню предназначено для выполнения часто используемых операций. Оно состоит из разделов, в которых могут быть подразделы. Каждый раздел состоит из конечного числа пунктов. Разделы и пункты имеют названия. Для быстрого доступа к пунктам меню часто используют горячие клавиши.

Всплывающие меню

Всплывающие меню имеют предназначение схожее с обычным. Оно скрыто от глаз пользователя и появляется только при щелчке правой кнопкой мыши. Всплывающие меню в большинстве случаев не имеет подразделов и горячих клавиш.

Переключатель

Переключатель похож на флажок, но позволяет выбрать только один пункт.

Надпись

Надпись представляет собой полосу произвольных размеров с текстом, которая расположена в заданном месте. Она так же предназначена для информирования пользователя.

Контейнер с полосами прокрутки

Контейнер с полосами прокрутки предназначен для хранения элементов, которые не вмещаются в окно. Полосы прокрутки бывают горизонтальные и вертикальные. Они позволяют прокручивать содержимое окна.

Поле ввода текста из многих строк

Оно является основным элементом любого текстового редактора и скорее всего вы с ним уже встречались.

Вкладки

Контейнер с вкладками фактически позволяет организовать несколько окон внутри одного. Так же как и окно, вкладка имеет заголовок. Вкладка не имеет меню. Переключение между вкладками осуществляется щелчком по заголовку.

Список значений

Список значений представляет собой квадрат, в котором находятся значения. Каждое значение представляет строку текста. Выбор значения осуществляется щелчком по элементу.

Раскрывающийся список

Раскрашивающийся список аналогичен по назначению списку значений, но позволяет хранить значения в более компактном виде. В мощных текстовых редакторах его используют для выбора гарнитуры шрифта.

Индикатор прогресса

Индикатор прогресса предназначен для того чтобы информировать о ходе выполнения каких-либо действий. Она представляет собой прямоугольную панель с постепенно удлиняющейся полосой определенного цвета.

Бегунок

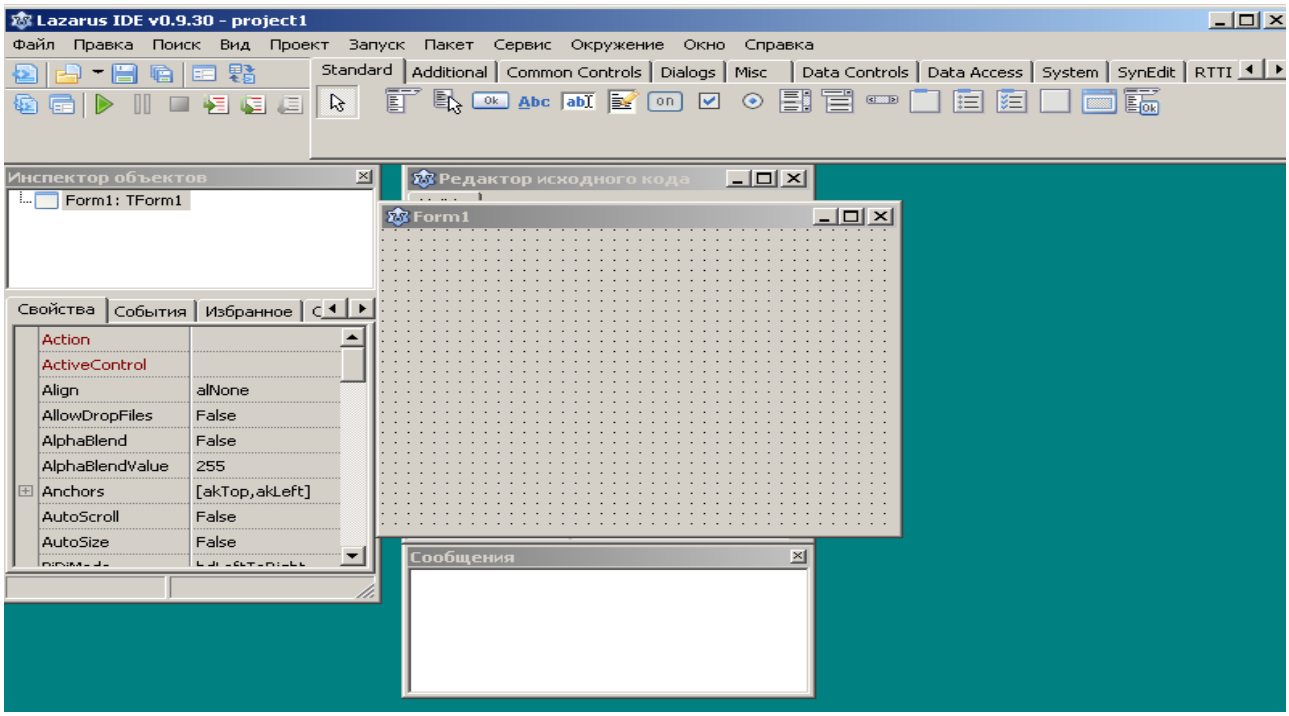
Бегунок позволяет выбрать значение из заданного диапазона. Бегунки бывают горизонтальные и вертикальные. Перемещение бегунка влево уменьшает значение на определенное число, а перемещение бегунка вправо увеличивает значение на определенное число.

Кнопка

Кнопка представляет собой квадрат с надписью. При щелчке на нем выполняется определенное действие.

При запуске Lazarus автоматически создает новый проект.

Вы увидите следующие окно:



Сразу под меню находиться палитра компонентов. В терминах Lazarus компонентами называют элементы интерфейса создаваемой программы.

Под палитрой компонентов находиться окно создаваемой программы, называемое формой. Слева от формы находиться инспектор объектов, который содержит свойства и события компонентов. Напомним, что компонент является объектом соответствующего класса. Имя объекта задается через свойство Name в инспекторе объектов.

Событием называется воздействие на программу пользователя или операционной системы.

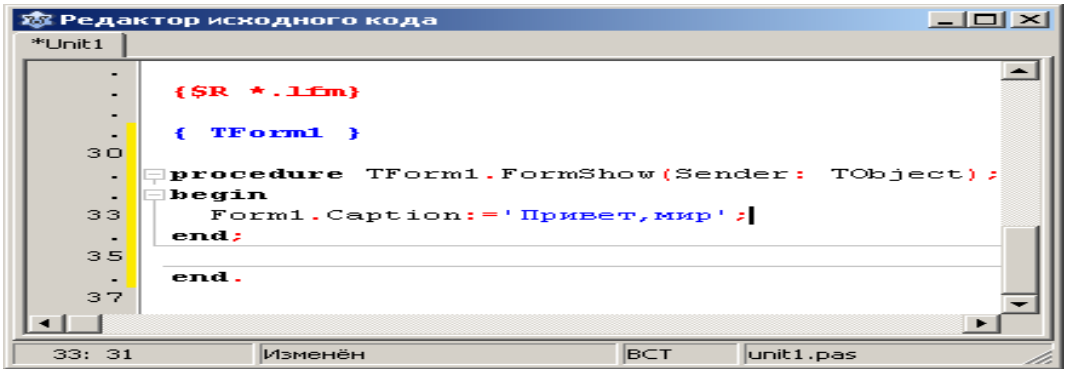
В качестве примера события можно назвать щелчок левой кнопки мыши по элементу интерфейса программы. Обработчики событий являются основой поведения программы.

Щелкните левой кнопкой мыши напротив нужного события на вкладке событий в инспекторе объектов, чтобы задать обработчик для компонента.

Откроется редактор исходного кода, который содержит шаблон обработчика события. Обработчик представляет собой метод класса.

Сам обработчик пишется в теле метода.

Пример обработчика события дан на рисунке ниже.



Глава 4. Основные элементы интерфейса

Окно

Каждая программа с графическим интерфейсом содержит как минимум одно окно, внутри которого отображаются остальные элементы интерфейса и необходимая информация. Часто окно имеет заголовок. Наиболее типичным заголовком является название программы. Окно является объектом класса TForm.

Методы

Метод	Аргументы	Возвращаемое значение	Описание
Show	Нет	Нет	Показывает окно на экране
ShowModal	Нет	Целое число	Показывает окно как модальное
Close	Нет	Нет	Закрывает окно

События

Событие	Описание
OnResize	Происходит при изменении размеров окна
OnShow	Происходит при появлении окна на экране
OnHide	Происходит при исчезновении окна

Свойства

Свойство	Тип	Описание
Caption	Строка	Текст в заголовке окна
Width	Целое число	Высота окна
Height	Целое число	Ширина окна
ClientWidth	Целое число	Высота клиентской области
ClientHeight	Целое число	Ширина клиентской области
BorderStyle	Список констант	Стиль границ окна
Font	Объект TFont	Шрифт элементов интерфейса

Кнопка

Кнопка является объектом класса TButton.

Свойства

Свойство	Тип	Описание
Top	Целое число	Y координата в окне
Left	Целое число	X координата в окне
Caption	Строка	Надпись на кнопке
Hint	Строка	Текст внутри всплывающей подсказки
ShowHint	Логическое значение	Наличие всплывающей подсказки
Visible	Логическое значение	Видимость кнопки на экране
Enable	Логическое значение	Определяет доступность кнопки

События

Событие	Описание
OnClick	Щелчок на кнопке
OnFocus	Получение фокуса

Надпись на форме

Надпись на форме является объектом класса TLabel. Свойства и события аналогичны TButton.

Переключатель и флажок

Переключатель является объектом класса TRadioButton. Флажок объектом класса TCheckBox. Они имеют общие свойства и события. Переключатели и флажки часто встречаются в окнах, отвечающих за настройку программ.

События

Событие	Описание
OnClick	Щелчок на кнопке
OnFocus	Получение фокуса

Свойства

Свойство	Тип	Описание
Caption	Строка	Текст на переключателе или флажке
Checked	Логическое значение	Определяет, выбран ли переключатель или флажок

Полоса статуса

Полоса статуса имеет длину равную ширине клиентской области и всегда находиться внизу окна. Полоса статуса является объектом класса TStatusBar. Текст, отображаемый в полосе статуса, определяется свойством SimpleText.

Поле ввода

Практически любая серьезная программа должна для выполнения своей задачи получить от пользователя необходимую информацию. Поле ввода позволяет пользователю ввести необходимую информацию. Для повышения надежности желательно перед обработкой введенных данных проверить их корректность. В случае ошибки можно предложить повторить ввод или скорректировать введенное значение. Поле ввода является объектом класса TEdit.

Свойства

Свойство	Тип	Описание
Text	Строка символов	Текст в поле ввода
ReadOnly	Логическое значение	Запрещает возможность редактирования
MaxLength	Целое число	Максимальная длина текста

События

Событие	Описание
OnChange	Изменение текста
OnEditingDone	Завершение ввода текста

Диалог выбора шрифта

Диалог выбора цвета является объектом класса TFontDialog. Вызов этого диалога происходит через метод Execute. Этот метод возвращает ложное логическое значение, если пользователь закрыл диалог и не выбрал шрифт. Выбранный шрифт содержится в свойстве Font.

Диалог выбора цвета

Диалог выбора цвета является объектом класса TColorDialog. Его вызов осуществляет метод Execute, который не берет аргументов. Он возвращает ложное логическое значение, если пользователь закрыл диалог и не выбрал цвет. Выбранный цвет содержится в свойстве Color.

Список значений

Список значений представляет собой объект класса TListBox.

Свойства

Свойство	Тип	Описание
MaxLength	Целое число	Максимальная длина элемента
Items	Массив объектов класса TStrings	Список значений
ItemIndex	Целое число	Индекс выбранного элемента

Раскрывающийся список

Раскрывающийся список является объектом класса TComboBox. Он позволяет выбрать нужный элемент из списка, представленного в компактном виде.

Свойства

Свойство	Тип	Описание
MaxLength	Целое число	Максимальная длина элемента
ItemIndex	Целое число	Индекс элемента
Text	Строка	Текст в поле ввода раскрывающегося списка
Sorted	Логическое значение	Определяет наличие сортировки элементов
Items	Объект класса TString	Представляет доступ к элементам списка

События

Событие	Описание
OnClick	Щелчок по списку
OnSelectionChange	Выбор элемента

Индикатор прогресса

Индикатор прогресса является объектом класса TProgressBar. Он полезен, если программа совершает операции, которые занимают много времени.

Свойства

Свойство	Тип	Описание
Max	Целое число	Максимальное значение прогресса
Min	Целое число	Минимальное значение прогресса
Position	Целое число	Текущее значение прогресса
Step	Целое число	Величина шага
Orientation	Константа	Ориентация индикатора прогресса
Smooth	Логическое значение	Истинное значение делает индикатор сплошным

Методы

Метод	Параметры	Возвращаемое значение	Описание
StepIt	Нет	Нет	Увеличивает значение прогресса
StepBy	Целое число	Нет	Увеличивает значение прогресса на заданное число шагов

Бегунок

Бегунок является объектом класса TTrackBar

Свойства

Свойство	Тип	Описание
Max	Целое число	Максимальное значение бегунка
Min	Целое число	Минимальное значение бегунка
Position	Целое число	Текущее значение бегунка
Orientation	Константа	Ориентация бегунка

Константы ориентации

Константа	Описание
trHorizontal	Горизонтальная ориентация
trVertical	Вертикальная ориентация

События

Событие	Описание
OnClick	Щелчок на бегунке
OnChange	Изменение позиции бегунка

Поле для ввода и редактирования текста из нескольких строк

Поле для ввода и редактирования текста из нескольких строк является объектом класса TMemo. Для загрузки текста из файла пользуйтесь методом LoadFromFile объекта Lines. Для сохранения текста в файл методом SaveToFile того же объекта. Оба метода берут в качестве параметра строку с именем файла.

Свойства

Свойство	Тип	Описание
WorldWrap	Логическое значение	Перенос текста по словам
MaxLength	Целое число	Максимальная длина одной строки
Lines	Объект типа TStrings	Дает доступ к строкам текста

Отображение изображений

Для отображения изображений используется компонент TImage. Он нужен, если в процессе своей работы, программа должна выводить изображения. Для загрузки изображения из файла или сохранения в файл воспользуйтесь свойством Picture. Оно является объектом класса TPicture. Для загрузки изображения из файла пользуйтесь методом LoadFromFile объекта Picture. Для сохранения изображения в файл методом SaveToFile того же объекта.
Оба метода берут в качестве аргумента строку с именем файла.

Свойства

Свойство	Тип	Описание
AutoSize	Логическое значение	Автоматическое изменение размера компонента
Center	Логическое значение	Отображение изображение по центру
Strech	Логическое значение	Подгонка изображения под размер компонента
Proportional	Логическое значение	Подгонка под размер компонента без искажения

События

Событие	Описание
OnPictureChanged	Происходит при изменении изображения
OnPaint	Происходит при рисовании изображения

Меню

Главное меню отображается наверху окна, а всплывающие вызывается по щелчку правой кнопкой мыши. Главное меню является объектом класса TMainMenu. Всплывающие меню является объектом класса TpopupMenu. Они оба используют массив объектов класса TMenuItems для хранения пунктов меню. Объект данного класса для определения названия пункта меню использует свойства строкового типа Caption. Для задания комбинации горячих клавиш используйте свойство ShortCut объектов класса TmenuItems.

Для присвоения ему значения воспользуйтесь функцией TextToShortcut.
Отследить активацию пункта меню можно при помощи события OnClick.

Чтобы привязать всплывающие меню к определенному компоненту воспользуйтесь свойством PopupMenu данного компонента.

Вкладки

Для размещения в окне нескольких вкладок используйте объект класса TPageControl. Вкладки позволяют лучше организовать отображение информации внутри одного окна. Вкладки, как и окна, имеют заголовок. Каждая вкладка является контейнером для элементов интерфейса.

Свойства

Свойство	Тип	Описание
ActivePageIndex	Целое число	Индекс активной вкладки
ActivePage	Объект класса TTabSheet	Дает доступ к активной вкладке
Pages	Массив объектов класса TTabSheet	Предоставляет доступ к вкладкам
PageCount	Целое число	Количество вкладок

События

Событие	Описание
OnChange	Переход на другую вкладку
OnChanging	Происходит во время перехода на другую вкладку

Чтобы задать заголовок вкладки воспользуйтесь строковым свойством Caption объекта класса TTabSheet.

Глава 5. Диалоги

Диалоги открытия и сохранения файла

Диалог открытия файла нужен, чтобы выбрать файл для открытия. Диалог сохранения используется, чтобы задать имя и расположения файла для сохранения данных. Диалог открытия файла является объектом класса TOpenDialog. Диалог сохранения является объектом класса TSaveDialog. Они имеют общие свойства и события. Диалоги вызываются методом Execute. Этот метод не берет аргументов и возвращает ложное логическое значение, если пользователь не выбрал файл.

Свойства

Свойство	Тип	Описание
Title	Строка	Текст в заголовке диалога
DefaultExt	Строка	Расширение файла по умолчанию
InitialDir	Строка	Каталог с файлами по умолчанию
FileName	Строка	Имя файла
Filter	Строка	Список расширений файлов
FilterIndex	Целое число	Номер выбранного фильтра

События

Событие	Описание
OnShow	Появление диалога на экране
OnClose	Закрытие диалога без выбора файла
OnCanClose	Закрытие диалога с выбором файла

Контейнер с полосами прокрутки

Контейнер с полосами прокрутки предназначен для размещения внутри него других элементов. Он является объектом класса TScrollBox. Логическое свойство AutoScroll скрывает или показывает полосы прокрутки.

Диалог выбора каталога

Диалог выбора каталога является объектом класса TSelectDirectoryDialog. Этот диалог вызываются методом Execute, Он не берет аргументов и возвращает ложное логическое значение, если пользователь не выбрал каталог.

События

Событие	Описание
OnShow	Появление диалога на экране
OnClose	Закрытие диалога без выбора каталога
OnCanClose	Закрытие диалога с выбором каталога

Свойства

Свойство	Тип	Описание
Title	Строка	Текст в заголовке диалога
InitialDir	Строка	Каталог по умолчанию
FileName	Строка	Имя каталога

Глава 6. Таймер

Таймер предназначен для выполнения кода через определенные интервалы времени. Это часто бывает полезным. Таймер является объектом класса Ttimer.

События

Событие	Описание
OnTimer	Срабатывание таймера
OnStartTimer	Включение таймера
OnEndTimer	Остановка таймера

Свойства

Свойство	Тип	Описание
Interval	Целое число	Интервал в миллисекундах
Enabled	Логическое значение	Активность таймера

Глава 7. Запуск программ

Для запуска программ используйте объект класса TProccess.

Методы

Метод	Параметры	Возвращаемое значение	Описание
Execute	Нет	Логическое значение	Запускает указанную программу
WaitOnExit	Нет	Логическое значение	Ждет завершения программы
Terminate	Нет	Логическое значение	Немедленно завершает программы
Suspend	Нет	Целое число	Приостанавливает программы
Resume	Нет	Целое число	Возобновляет работу программы

Константы приоритета

Константа	Описание
ppHight	Высокий приоритет
ppIdle	Запуск только при не активности системы
ppNormal	Нормальный приоритет
ppRealTime	Режим реального времени

Константы управления окном терминала

Константа	Описание
swoNone	Окном управляет операционная система
swoHIDE	Главное окно скрыто
swoMaximize	Главное окно раскрывается на полный экран
swoMinimize	Главное окно будет свернуто
swoRestore	Восстановление предыдущей позиции
swoShow	Будет показано главное окно
swoShowDefault	Задает использование параметров по умолчанию

Свойства

Свойство	Тип	Описание
Active	Логическое значение	Запускает или останавливает процесс
ApplicationName	Строка	Имя программы для запуска
CommandLine	Строка	Параметры командной строки
ConsoleTitle	Строка	Заголовок окна терминала
CurrentDirectory	Строка	Текущий каталог
Executable	Строка	Имя исполняемого файла
Priority	Список констант	Приоритет процесса
ShowWindow	Список констант	Управление окном терминала
ExitStatus	Целое число	Содержит код возврата программы.
Running	Логическое значение	Определяет, запущен ли процесс
ProccessID	Целое число	Идентификатор процесса

Заключение

Закончена книга, но не закончено ваше изучение Lazarus. Чтобы стать хорошим программистом необходимо много практики. Кроме того нужно уметь читать официальную документацию. В ней вы найдете ответы на интересующие вопросы, так как документация достаточно подробна и понятно написана.

В рамках данного справочника невозможно охватить все темы, касающиеся языка Free Pascal и среды Lazarus. Поэтому вам необходимо будет самостоятельно углублять сведения, полученные из этой книги.

Вокруг рассмотренного в этой книге языка и среды сложилось обширное сообщество. В интернете вы найдете множество полезной информации по Free Pascal и Lazarus. Большинство найденной информации будет на англоязычных сайтах. Это связано с целевой аудиторией использующей Free Pascal и Lazarus. Данные продукты завоевали широкую популярность среди иностранных программистов благодаря открытости и высокому качеству.

Free Pascal не получил широкого распространения в России из-за доминирующей позиции Delphi. Однако в России тоже есть сообщество пользователей Free Pascal. Оно менее многочисленно, чем иностранное, но там вы всегда найдете людей способных помочь в освоении Free Pascal. Русскоязычное сообщество имеет свой сайт в интернете. На нем вы найдете полезные статьи по Free Pascal и Lazarus и активный форум. Адрес сайта — http://freepascal.ru/

В этой книге я постарался изложить основные сведения необходимые вам для старта. Надеюсь, он будет успешным. А пока я прощаюсь с читателем и желаю ему удачи.

Список литературы

1. Официальная документация по Free Pascal - http://sourceforge.net/projects/freepascal/files/Documentation/
2. Официальная документация по Lazarus - http://sourceforge.net/projects/lazarus/files/Lazarus%20Documentation/
3. Free Pascal и Lazarus: Учебник по программированию - http://www.altlinux.org/Books:FreePascal
4. Основы программирования в среде Lazarus - http://mansurov-oshtu.ucoz.ru/
5. В.В.Фаронов. ОСНОВЫ ТУРБО ПАСКАЛЯ, СП УИЦ «МВТУ-ФЕСТО ДЕДАКТИК», Москва 1991