Е.А.Попов

Экспресс курс программирования в Lazarus

Сорок пятая редакция

2011 — 2016 год

Содержание

Введение

Данная книга представляет собой краткий справочник, содержащий необходимые сведения для того чтобы освоить один из вариантов языка Паскаль 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 со многолетней практикой программирования. Причиной перехода на Lazarus явилась 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 рассчитан на Free Pascal версии 3.0.0 и среду Lazarus версии 1.6. Однако изложенные сведения применимы и к более новым версиям.

Условия распространения книги и обратная связь с автором

 Связаться с ним можно послав письмо по электронной почте на адрес tuzik87@inbox.ru

Часть 1. Основные сведения о диалекте Free Pascal

Глава 1. Идентификаторы и ключевые слова

Идентификаторы

Идентификаторы используются в качестве имен для переменных и других элементов программы. Каждый идентификатор состоит из букв английского алфавита. В них так же могут встречаться цифры. Паскаль не делает разницы между большими и строчными буквами. Каждый идентификатор начинается с буквы или символа подчеркивания. Минимальная длина идентификатора составляет один символ. Максимальная длина идентификатора в данной реализации Паскаля равна 127 символов.

Ключевые слова

Ключевые слова используются для построения языковых конструкций. Список ключевых слов дан ниже.

Ключевые слова Free Pascal:

dispose, exit, new

Ключевые слова, заимствованные из Turbo Pascal:

absolute, and, array, asm, begin, case, const, constructor, destructor, div, do, downto, else, end, file, for, function, goto, if, implementation, in, inherited, inline, interface, label, mod, nil, not, object, of, on, operator, or, packed, procedure, program, record, reintroduce, repeat, self, set, shl, shr, string, then, to, type, unit, until, uses, var, while, with, xor

Ключевые слова, заимствованные из Object Pascal:

as, class, dispinterface, except, exports, finalization, finally, initialization, inline, is, library, on, out, packed, property, raise, resourcestring, threadvar, try

Модификаторы:

absolute, abstract, alias, assembler, cdecl, cppdecl, default, export,
external, far, far16, forward, index, local, name, near, nostackframe
oldfpccall, override, pascal, private, protected, public, published,
read, register, reintroduce, safecall, softfloat, stdcall, virtual, write

Глава 2. Конфликты идентификаторов

Использование идентификатора, который совпадает с ключевым словом, приведет к ошибке компиляции. Для разрешения конфликта поставьте знак амперсанда(&) перед этим нестандартным идентификатором. Амперсанд необходим при каждом использовании нестандартного идентификатора.

Глава 3. 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.

Привязка данных

Двум переменным можно назначить одну и ту же область памяти.

Синтаксис: var имя: тип absolute цель;

Целевая переменная должна быть заранее объявлена и не должна быть инициализирована. Инициализацией называют присвоение начального значения.

Константы

Константы отличаются от переменных тем, что не могут изменять значение. Константа объявляется при помощи следующей конструкции:

const имя=значение;

Типизированные константы

В обычных константах тип хранимых данных определяется автоматически. В типизированных константах он задается явно.

Синтаксис: const имя:тип=значение;

Целые типы

Целые типы делятся на знаковые и типы без знака.

Тип	Описание	Диапазон	Размер в байтах
Byte	Байт	От 0 до 255	1
shortint	Целое число со знаком	От -128 до 127	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

Вещественные типы

Есть несколько вещественных типов. Некоторые из этих типов имеют специфические особенности.

Тип Comp поддерживается не на всех платформах. Тип Real не имеет двоичной совместимости с аналогичным типом из Turbo Pascal.

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

Логические типы

Для хранения булевых значений в данной реализации Паскаля есть четыре типа данных.

Тип	Размер в байтах	Внутренние представление истины
Boolean	1	1
ByteBool	1	Ненулевое положительное значение
WordBool	2	Ненулевое положительное значение
LongBool	4	Ненулевое положительное значение

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.

Использование шестнадцатеричных, восьмеричных и двоичных чисел

Шестнадцатеричному числу должен предшествовать символ $. Восьмеричному числу предшествует символ &, а двоичному числу символ %. Пробел между символом и числом недопустим.

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 однотипных констант удобно объединить в список, называемый перечислением. Перечисление определяется при помощи следующей конструкции:

Type перечисление=(константа1:=значение1,...,константаN:=значениеN);

Перечисление является одним из самостоятельно определяемых типов данных. Присваивание константе начального значения не является обязательным. В этом случае ее значение будет на единицу больше предыдущей константы. По умолчанию первая константа равна нулю.

Псевдонимы типов

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

Синтаксис: Type псевдоним=тип;

Кроме встроенных типов псевдонимы можно также создавать для массивов и указателей, а так же файловых типов.

Ограничение диапазона

Вы можете ограничить диапазон значений уже существующего типа данных.

Синтаксис: Type тип=минимум..максимум;

Явное преобразование типов

Преобразование между большинством встроенных типов данных выполняется автоматически. Выполнить явное преобразование можно при помощи следующей конструкции: переменная:=тип(цель);

Глава 4. Области видимости данных

Глобальные данные объявляются вне блоков и доступны везде. Память под глобальные данные выделяется при старте и освобождается при завершении программы. Глобальные данные хранятся в специальной области.

Локальные данные доступны только в своем блоке. Память под локальные данные выделяется при входе в соответствующий блок и освобождается при выходе из него. Выделение происходит из стека.

Глава 5. 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.

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.

Одиночный оператор так же является выражением.

Блочный оператор

Оператор, объединяющий в себе другие, называется блочным.

Синтаксис:
begin
операторы
end;

Оператор goto

Оператор goto выполняет переход к указанной метке. Этот оператор действует только в пределах своего блока.

Метка 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

Представлен широкий набор математических операторов. Они могут быть использованы в выражениях для выполнения математических вычислений.

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

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.

Оператор возведения в степень работает только с целыми положительными переменными.

Комбинированные операторы

Данная реализация Паскаля располагает набором комбинированных операторов. Они позаимствованы из языка Си. Комбинированные операторы являются бинарными.

Они работают следующим образом. Вначале выполняется арифметическая операция с использованием левого и правого операндов, а затем результат вычислений присваивается левому операнду.

Оператор	Описание
+=	Сложение с присваиванием
-=	Вычитание с присваиванием
*=	Умножение с присваиванием
/=	Деление с присваиванием

Операторы сравнения

Все операторы сравнения являются бинарными. Обычно они используются вместе с условными операторами и циклами. Об условных операторах и циклах вы узнаете чуть позже из этой книги.

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

Логические операторы

В булевой алгебре выделяют три основные операции: конъюнкция, дизъюнкция и инверсия. Конъюнкция и дизъюнкция являются бинарными, а инверсия является унарной операцией.

Инверсия изменяет значение аргумента на противоположенное. Конъюнкция возвращает истинное значение, когда оба аргумента истинны, а дизъюнкция возвращает истинное значение, когда один из аргументов имеет истинное значение.

Оператор «Логическое Не» выполняет инверсию. Оператор «Логическое И» выполняет конъюнкцию. Оператор «Логическое Или» выполняет дизъюнкцию.

Оператор	Описание	Вид оператора
not	Логическое Не	Унарный
and	Логическое И	Бинарный
or	Логическое Или	Бинарный

Поразрядные логические операторы

Поразрядные логические операторы отличаются от обычных тем, что работают с каждым отдельным битом переменной. Отдельного внимания заслуживает битовый сдвиг и «Исключающие Или».

«Исключающие Или» дает на выходе единицу если поданные на вход аргументы не равны друг другу.

При выполнении сдвига часть бит сдвигается на определенную величину. Выполнение сдвига влево дает результат аналогичный умножению числа на степень двойки.

Выполнение сдвига вправо дает результат аналогичный делению числа на степень двойки.

Оператор	Описание
not	Побитовое логическое Не
and	Побитовое логическое И
or	Побитовое логическое Или
Xor	Исключающие Или
Shl	Побитовый сдвиг влево
Shr	Побитовый сдвиг вправо
>>	Побитовый сдвиг влево
<<	Побитовый сдвиг вправо

Глава 6. 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 then должна отсутствовать точка с запятой после ключевого слова end. Блочный оператор, идущий после ключевого слова else, 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 7. 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. Синтаксис этого оператора имеет две формы.

Первая форма: for переменная:=начало to конец do оператор;

Вначале определяется начальное и конечное значение переменной. Значение переменной увеличивается на единицу в конце каждой итерации цикла.

Вторая форма: for переменная:=начало downto конец do оператор;

Во второй форме значение переменной уменьшается на единицу в конце каждой итерации цикла.

Цикл for..in..do

Оператор for..in..do проходит по всем элементам множества. При этом переменная содержит текущий элемент. Оператор также может применяться к массивам и строкам.

Синтаксис: for переменная in множество do оператор;

Цикл while..do

Цикл while..do выполняет оператор пока условие истинно. Проверка условия происходит перед выполнением оператора.

Синтаксис: while условие do оператор;

Цикл repeat..until

Цикл repeat...until выполняет оператор пока условие ложно. Условие проверяется после выполнения 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.

Управление циклом

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

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

Глава 8. 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 состоит из операторов. Телу 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 определенного типа.

Вызов подпрограмм

Вызов подпрограммы осуществляется через ее имя. Для вызова подпрограммы используйте следующую конструкцию: подпрограмма(аргументы);

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.

Альтернативный вызов подпрограмм

Вы можете воспользоваться альтернативным вариантом вызова подпрограммы, если ей не нужно передавать аргументы.

Формат вызова подпрограммы: подпрограмма;

Определение процедуры

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

```
procedure подпрограмма(параметр1;...параметрN);
Объявления
begin
тело
end;
```

Определение функции

Функция определяется следующим образом:

function функция(параметр1;...параметрN):тип;
Объявления
begin
тело
end;

Напишите следующую конструкцию в конце тела функции, чтобы возвратить значение: функция:=значение;

В режиме совместимости с Object Pascal или Delphi имя функции может быть заменено ключевым словом Result.

В режиме совместимости с Object Pascal вы так же можете вызвать процедуру Exit и передать ей некоторое значение в качестве аргумента. Это приведет к немедленному выходу из вашей подпрограммы и возврату нужного значения. Вызов процедуру Exit может располагаться в любом месте тела 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 доступен только внутри подпрограммы. Переданные при вызове подпрограммы аргументы присваиваются соответствующим параметрам.

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. Параметры по умолчанию должны находиться в конце списка параметров.

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. Выходные параметры не могут быть использованы в правой части выражений.

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. У не типизированных параметров отсутствует указание типа. В качестве не типизированных параметров могут выступать параметры-переменные, постоянные и выходные параметры.

Перегрузка подпрограмм

Одно имя может соответствовать нескольким подпрограммам. Это называется перегрузкой. Перегрузка 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 9. Математика

Стандартные математические подпрограммы

В распоряжение программиста доступен широкий набор математических подпрограмм. В таблице ниже перечислены основные математические подпрограммы.

Подпрограмма	Описание
odd(число)	Возвращает истину если число является четным
Abs(число)	Модуль числа
Sqr(число)	Квадрат числа
Sqrt(число)	Квадратный корень числа
Round(число)	Округление дроби до ближайшего целого
Trunc(число)	Округление дроби в меньшую сторону
Randomize	Инициализация генератора случайных чисел
Random(число)	Возвращает случайное значение в диапазоне от 0 до число-1
Cos(угол)	Косинус
Sin(угол)	Синус
tan(угол)	Тангенс
pi()	Возвращает значение числа Пи

Управление генератором случайных чисел

Зерном называют число используемое генератором случайных чисел для генерации случайных значений. Заданное по умолчанию зерно можно изменить используя предопределенную переменную RandSeed.

Глава 10. Модификаторы подпрограмм

Общие сведения

Модификаторы могут быть использованы в определениях подпрограмм. Модификатор указывается сразу после имени подпрограммы после точки с запятой. В конце модификатора так же ставится точка с запятой. Использование модификаторов не является обязательным.

Игнорируемые модификаторы

Компилятор игнорирует модификаторы far, far16 и near. Эти модификаторы нужны только для совместимости с диалектом Turbo Pascal.

Обработчики файлов

Модификатор iocheck указывает на то, что функция является обработчиком файловых операций и может возвращать код ошибки.

Еще раз о перегрузке подпрограмм

Для перегружаемых подпрограмм вы можете использовать модификатор overload, который введен для совместимости с Delphi.

Использование модификатора overload обязательно, только если подпрограммы расположены в другом модуле.

Встраиваемые подпрограммы

Для встраиваемых подпрограмм используется модификатор inline. Для таких подпрограмм вызов заменяется вставкой кода из тела подпрограммы.

Ограничения встраиваемых подпрограмм:

1. Недопустима рекурсия и вызов других подпрограмм
2. Модификатор inline носит рекомендательный характер и при отсутствии возможности использования встраиваемых подпрограмм, компилятор генерирует обычные подпрограммы

Псевдонимы подпрограммы

Вы можете определить дополнительное имя подпрограммы, которое допускается использовать вместо основного при вызове подпрограмм. Это имя называют псевдонимом. Псевдоним задается модификатором alias. Модификатор и псевдоним отделяются друг от друга пробелом.

Предварительные объявления

Предварительные объявления используются для описания подпрограмм, чье определение будет дано позже. Само определение подпрограммы делается обычным образом.

Предварительное объявление делается при помощи прототипа подпрограммы с модификатором forward. Прототипы подпрограмм отличаются от определений отсутствием основной части.

Внешние подпрограммы

Модификатор external указывает на то, что подпрограмма будет находиться во внешнем объектном файле. Модификатор export указывает на то, что подпрограмма может быть экспортирована в подключаемую библиотеку.

Программная эмуляция математического сопроцессора на процессорах ARM

Используйте модификатор softfloat чтобы задействовать программную эмуляцию математического сопроцессора на процессорах ARM. Математический сопроцессор применяется для обработки чисел с плавающей точкой.

Модификаторы, которые влияют на передачу аргументов в подпрограмму

Существует ряд модификаторов, влияющих на внутренний механизм передачи аргументов в подпрограммы.

Модификатор	Описание
cdecl	Стиль вызова Си. Аргументы помещаются в стек справа налево. Очистку стека выполняет вызывающая подпрограмма.
oldfpccall	Старый стиль вызова Free Pascal
pascal	Стиль вызова Паскаль. Аргументы помещаются в стек слева направо. Очистку стека выполняет вызванная подпрограмма.
stdcall	Аргументы помещаются в стек справа налево. Очистку стека выполняет вызванная подпрограмма.
safecall	Аргументы помещаются в стек справа налево. Очистку стека выполняет вызванная подпрограмма. Состояние регистров сохраняется перед вызовом и восстанавливается после выхода из подпрограммы.
register	Аргументы передаются через регистры процессора.
saveregisters	Аргументы передаются через регистры процессора. Состояние регистров сохраняется перед вызовом подпрограммы.
interrupt	Аргументы передаются через регистры процессора. Состояние регистров восстанавливается после выхода из подпрограммы.
nostackframe	Запрещает использовать стек для функции.
vargs	Используется для подпрограмм с переменным числом аргументов.

Модификаторы, определяющие область видимости

Модификатор	Описание
local	Локальные подпрограммы не могут быть экспортированы из библиотек, но позволяют обойтись без прототипов, если подпрограмма расположена в модуле.
public	Публичные подпрограммы недоступны в модуле, но могут вызываться из объектных файлов.

Глава 11. Указательные подсказки

Указательные подсказки являются разновидностью модификаторов. Ими можно маркировать идентификаторы, которые взаимодействуют с нестабильным кодом. Тогда при обработке этих идентификаторов во время компиляции будут выведены соответствующие предупреждающие сообщения.

Маркировать можно не только переменные и константы, но также подпрограммы и модули.

Подпрограммы маркируются обычным образом.

Замените точку с запятой в конце объявления на пробел с последующей указательной директивой, чтобы маркировать константы и переменные. В конце конструкции поставьте точку с запятой.

Аналогично маркируются имена модулей.

Список подсказок дан в таблице ниже.

Подсказка	Описание
deprecated	Устаревший код
experimental	Экспериментальный код
platform	Код специфичный для конкретной системы
unimplemented	Только для подпрограмм. Не реализованная подпрограмма

Глава 12. Приращение и уменьшение

Подпрограмма Inc увеличивает значение переменной на указанную величину. Подпрограмма Dec выполняет обратное действие. Они берут имя переменной и значение величины в качестве аргументов.

Эти подпрограммы работают только с целочисленными переменными. Второй аргумент является необязательным и может пропущен. В этом случае переменная будет уменьшена или увеличена на единицу.

Глава 13. Консоль

Что такое консоль?

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

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.

Ввод заканчивается нажатием клавиши Enter. Переменным присваиваются введенные значения.

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.

Обе процедуры берут имя переменной или выражение в качестве аргумента. Он заменяется списком аргументов, если необходимо ввести несколько значений. Аргументы в списке отделяются запятой.

Форматированный вывод

Передайте подпрограмме вывода дополнительный двойной спецификатор чтобы вывести на консоль вещественное число как число с фиксированной запятой.

Он состоит из двух частей разделенных двоеточием: максимальное количество выводимых символов и количество цифр в дробной части.

Спецификатор добавляется к переменной, которая передается подпрограмме консольного вывода в качестве аргумента. Переменная отделяется от спецификатора двоеточием.

Чтобы ограничить число выводимых символов для целого числа или строки используйте одиночный спецификатор.

Глава 14. 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 таблицу чисел. Математики называют таблицу чисел матрицей.

Объявление массива

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

Размерность указывает количество элементов в массиве. Размерность задается как номер первого и номер последнего элемента, отделенные друг от друга двумя точками. В многомерном массиве следующая размерность указывается после запятой.

Инициализация массивов при объявлении

Чтобы инициализировать массив при объявлении поставьте вместо точки с запятой следующую конструкцию: :=(список);

Элементы в списке разделяются 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 многомерного массива отличается от объявления одномерного массива.

Пример объявления одномерного динамического массива:
var simple: array of Byte;
Пример объявления двумерного динамического массива:
var matrix: array of array of Byte;

Вызов подпрограммы SetLength выделяет место под массив в динамической памяти.

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

Размерность задается, как число определяющее количество элементов. В многомерном массиве размерность заменяют на список размерностей. Элементы списка отделяются друг от друга запятой.

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

Псевдонимы массивов

Предусмотрено две конструкции для определения псевдонимов массивов.

Вариант для обычного массива: type псевдоним=array[размерность] of тип;
Вариант для динамического массива: type псевдоним=array of тип;

Эти псевдонимы могут быть использованы позже при объявлении массива.

Конструктор динамического массива

Компилятор автоматически создает для каждого динамического массива специальную подпрограмму. Она выделяет память под массив и заполняет его данными. Эту подпрограмму называют конструктором динамического массива. Конструктор имеет имя Create.

Формат вызова конструктора: массив:=псевдоним.Create(аргументы);

Нумерация в динамических массивах

В динамических массивах нумерация подчиняется жестким правилам. Первый элемент имеет номер 0. Номер последнего элемента на единицу меньше размерности массива.

Границы массивов

Выход за границы массива может привести к повреждению данных. Также он делает вашу программу неустойчивой и позволяет внедрить и выполнить внутри вашей программы посторонний код.

Границы одномерных массивов

Для определения границ одномерных массивов предусмотрено две функции. Функция Low возвращает номер первого элемента, а функция High возвращает номер последнего элемента массива. Обе функции берут имя массива в качестве аргумента.

Глава 15. Указатели

Предназначение указателей

Указатель является переменной, которая хранит адрес участка памяти. Все указатели имеют фиксированный размер, который зависит от платформы и равен размеру машинного слова.

Присваивание указателям значения

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

Операция разыменования

Эта операция противоположена по смыслу операции взятия адреса. Операция разыменования предназначена для получения доступа к данным, на которые ссылается указатель.

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;

Особенности типизированных указателей

Типизированные указатели имеют свои особенности.
Разность указателей возвращает количество элементов расположенных между адресами. Указатели так же могут быть операндами в операторах сравнения.

Сдвиг адреса в типизированных указателях

Сдвинуть адрес внутри указателя можно двумя способами.

Можно прибавить или вычесть из указателя положительное целое значение. Величина сдвига представляет собой произведение целого значения на размер соответствующего типа данных.

Тот же результат дает использование подпрограмм Inc или Dec.

Типизированные указатели и массивы как аргументы подпрограмм

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

Глава 16. Динамическое распределение памяти

Что такое динамическая память?

Динамической называют память, которая может выделяться во время работы программы по желанию программиста. Динамическая память выделяется из кучи.

Особенности работы с блоками памяти

Работа с блоками памяти осуществляется через не типизированные указатели.

Выделение и освобождение блоков памяти

Для выделения блока памяти предусмотрено три подпрограммы.

Подпрограмма GetMem 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.

Подпрограмма AllocMem выделяет блок памяти и заполняет его нулевыми байтами. Она берет размер блока памяти в качестве аргумента и возвращает указатель на этот блок.

Подпрограмма ReAllocMem выделяет новый блок памяти и копирует в него данные из уже существующего блока. Она берет два аргумента: указатель на существующий блок памяти и размер нового блока памяти. Копирование данных будет произведено не полностью если размер нового блока меньше размера старого блока. Копирование данных не производится если передать nil в качестве первого аргумента. Подпрограмма ReAllocMem возвращает указатель на новый блок.

Освобождение блока памяти

Освобождение выделенного блока производится 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 предопределенной глобальной переменной ReturnNilIfGrowHeapFails значение True, чтобы обнулять указатель при нехватке памяти.

Сущности конкретного типа и динамическая память

В качестве примера таких сущностей можно назвать экземпляры записей и объектов без класса. Для работы с ними предусмотрено две подпрограммы.

Подпрограммы New и Dispose работают с типизированными указателями. Подпрограмма New выделяет память, а подпрограмма Dispose освобождает выделенную память.

Обе подпрограммы берут типизированный указатель в качестве аргумента.

Полезные подпрограммы для работы с памятью

В таблице ниже перечислены некоторые полезные подпрограммы для работы с динамической памятью.

Подпрограмма	Описание
MemSize(блок)	Возвращает размер блока памяти
SizeOf(элемент)	Возвращает размер элемента
addr(аргумент)	Возвращает адрес аргумента

Глава 17. Процедурный тип

Указатели на подпрограммы

Прежде чем использовать указатель на подпрограмму нужно определить новый тип данных. Этот тип называется процедурным.
Определение для процедуры: type тип=procedure(параметры);
Определение для функции: type тип=function(параметры):тип;

Объявление указателя на подпрограмму выполняется обычным образом.

Присваивание указателям адреса и вызов подпрограмм

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

Глава 18. Множества

Понятие множества и операции над ними

Понятие множества в Паскале аналогично тому, что принято в математике. Над множествами можно совершать ряд стандартных операций. К ним относятся: объединение, пересечение, разность, симметричная разность.

Операция объединения возвращает новое множество, состоящее из всех элементов обоих множеств.

Пересечение состоит из элементов принадлежащим обоим парам множеств.
Разность состоит из элементов первого множества, которые не входят во второе.

Симметричная разность дает множество состоящие из элементов, которые не являются общими для исходных множеств.

Создание экземпляра множества

Экземпляр множества создается аналогично обычной переменной. При создании множества нужно использовать один из порядковых типов. К этим типам относятся целые и булевы типы, а также символьный тип.

Синтаксис: var имя: set of тип;

Заполнение множеств

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

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

Ограничения множеств

Множество, не содержащее элементов, называют пустым. Максимальный размер множества, в данной реализации Паскаля, 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);

Операторы для работы со множествами

Для работы со множествами есть несколько операторов. Все они являются бинарными.

Оператор	Описание
+	Объединение
*	Пересечение
-	Разность
><	Симметричная разность
in	Проверка элемента на принадлежность множества

Проход по элементам множества

Функция Pred возвращает предыдущий элемент множества, а функция Succ возвращает следующий элемент множества. Обе функции берут имя множества в качестве аргумента. Функции Pred и Succ также применимы к перечислениям и массивам.

Глава 20. Обработка строк

Хранение строк и символов

Строка представляет собой последовательность символов. Есть несколько типов данных для хранения символов и строк.

Строковые типы данных

Строки могут быть представлены в памяти различными способами.

Тип	Описание
ShortString	Короткая строка
AnsiString	Обычная строка
RawByteString	Обычная строка без указания кодировки
String	Синоним ShortString или AnsiString
PChar	Строка с нулевым байтом в конце. Символ занимает один байт.
WideString	Широкая строка. Символ занимает два байта.
UnicodeString	Широкая строка, которая содержит внутри себя счетчик ссылок

Символьные типы данных

Предусмотрено три символьных типа.

Тип	Описание
Char	Символ, занимающий в памяти один байт
WideChar	Символ, занимающий в памяти два байта
AnsiChar	Синоним типа Char

Внутренние представление строк

Начнем со строк, где каждый символ занимает один байт.

Тип ShortString представляет собой короткую строку. Ее максимальная длина составляет 255 символов. Этот тип строк реализован как массив. Размер строки хранится в первом элементе. Этот тип введен для совместимости с Turbo Pascal.

Остальные строки не имеют ограничения по длине.

Тип PChar представляет собой массив символов с нулевым байтом на конце. Он введен, чтобы обеспечить возможность взаимодействие с кодом на языке Си или Си++.

Тип AnsiString похож по внутреннему устройству на PChar, но каждая строка кроме массива символов содержит внутри служебную информацию. Эта информация включает в себя длину строки и счетчик ссылок.

Теперь перейдем к строкам с размером символа больше одного байта.

Тип UnicodeString похож на AnsiString, но каждый символ в строке занимает 2 байта. Его использование является предпочтительным при работе с широкими строками.

Тип WideString похож на UnicodeString, но в строках этого типа отсутствует счетчик ссылок. Этот тип необходим при работе с COM объектами в системе Windows.

Выделение памяти под строки

Память под строки выделяется автоматически. Освобождение памяти происходит, когда строка не больше нужна.

Присваивание значений

Значение, которое присваивается строковой переменной, должно быть заключено в одинарные кавычки. Также допустимо присваивание значений, которые возвращают функции или присваивание значений другой строковой переменной.

Подпрограммы для работы с символами

Для работы с символами предусмотрено две функции. Функция ord возвращает код символа. Функция chr возвращает символ, который соответствует заданному коду. Функция chr берет коды в диапазоне от 0 до 255.

Склейка строк

Для склейки нескольких строк в одну используйте оператор суммирования +.

Ограничение длины коротких строк

Вы можете задать длину строковой переменной при ее объявлении.

Синтаксис: var имя:string[длина];

Преобразования кодировок

Функция AnsiToUtf8 преобразует строку из текущей системной кодировки в UTF-8. Функция Utf8ToAnsi выполняет обратное преобразование. Обе функции работают со строками типа RawByteString.

Процедуры преобразования строк

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

Следующие подпрограммы являются перегружаемыми и работают со всеми типами строк.

Подпрограмма	Описание
Length(строка)	Возвращает длину строки
SetLength(строка,длина)	Изменяет длину строки
LowerCase(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,строка)	Возвращает позицию, в которой находится подстрока
Delete(строка, позиция, количество)	Удаляет часть символов из строки

Глава 21. Записи

Запись как пользовательский тип данных

Запись представляет собой сущность, которая содержит и объединяет в себе разные переменные. Переменные внутри записи называют полями.

Описание записи

Синтаксис описания записи:

```
Type запись=record
поля
End;
```

Формат объявления поля: поле:тип;

Замените поле списком полей, чтобы объявить несколько однотипных полей в одной строке. Имена в списке разделяются запятой.

Объявление экземпляра записи и доступ к отдельным полям

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

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 использовать ее вместо стандартного типа в списке параметров. Так же запись может являться значением, которое возвращает функция.

Записи с вариантами

Запись может иметь вариантную часть, в которой находятся наборы необязательных полей. Обращение ко всем полям происходит обычным образом.

Доступность того или иного набора определяется текущим значением одного из служебных полей.

Синтаксис:
Type запись=Record
поля
case поле:тип of:
значение1:(набор1);
…
значениеN:(наборN);
End;

Объявление полей в наборах аналогично объявлению обычных полей. Поля разного типа отделяются друг от друга точкой с запятой.

Глава 22. Значения по умолчанию

Задание значений по умолчанию

Компилятор может автоматически присвоить переменной значение по умолчанию, основываясь на типе переменной. Делается это при помощи следующей конструкции: переменная:=default(тип);

Данная конструкция работает с базовыми типами и записями.

Правила задания значений по умолчанию для базовых типов

Правила задания значений по умолчанию для базовых типов приведены в таблице ниже.

Тип	Значение
Логические переменные	False
Целые числа со знаком и без знака	0
Числа с фиксированной и плавающей точкой	0.0
Символы	Пустой символ
Строки	Пустая строка
Указатели	Nil

Задание значений по умолчанию для структур

При задании значений по умолчанию для экземпляра структуры, ее полям присваиваются значения по умолчанию. Значение конкретного поля зависит от его типа.

Глава 23. Упакованные сущности

Выравнивание

Центральный процессор обрабатывает данные, используя машинные слова. Выравнивание позволяет оптимизировать размещение данных в памяти. Побочным эффектом выравнивания является то, что структуры данных могут занимать больше памяти, чем ожидалось.

Упаковка записей

Чтобы использовать упакованную запись нужно в ее описании перед ключевым словом record написать ключевое слово packed или bitpacked. При использовании ключевого слова bitpacked выравнивание выполняется с учетом отдельных бит.

Упаковка массивов

Чтобы использовать упакованный массив нужно в его объявлении перед ключевым словом array написать ключевое слово packed или bitpacked. Динамический массив не может быть упакованным.

Процедура pack упаковывает содержимое обычного массива и помещает содержимое в упакованный массив. Процедура unpack распаковывает содержимое упакованного массива и помещает содержимое в обычный массив.

Формат процедуры pack: pack(источник,позиция,цель);
Формат процедуры unpack: unpack(источник,цель,позиция);

Глава 24. Перегрузка операторов

Необходимость перегрузки операторов

При работе с записями бывает необходимо переопределить поведение операторов. Этот процесс называется перегрузкой операторов.

Реализация перегрузки

Перегрузка операторов реализуется при помощи следующей конструкции:

operator оператор(параметры) результат: тип;
begin
тело
end;

Эта конструкция напоминает определение подпрограммы. Параметры задаются так же как параметры подпрограмм.

Результат является локальной переменной, которая доступна в теле оператора. Значение результата будет возвращено оператором после выполнения работы.

В режиме совместимости с Object Pascal или Delphi вы можете использовать в теле оператора ключевое слово Result вместо результирующей переменной.

Глава 25. 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 Variant. Переменным типа Variant можно присваивать целочисленные и дробные переменные, а так же строки и интерфейсы.

Глава 26. Файлы

Файловые типы

Файлы предназначены для хранения данных. В Паскале для работы с файлами используются переменные специального типа. Этот тип называют файловым.

Типы файлов

Файлы бывают следующих типов: текстовые, типизированные и не типизированные. Типизированные файлы содержат блоки заданного типа.

Особенности работы с не текстовыми файлами

Размер блока не типизированного файла по умолчанию составляет 128 байт.
Размер блока типизированного файла зависит от типа блока.

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 не типизированного файла: var имя:file;

Типизированные файлы как аргументы подпрограмм

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

Общие подпрограммы

Начнем с общих подпрограмм, которые работают со всеми файлами.

Подпрограмма	Описание
Assign(файл,имя)	Связывает файловую переменную с конкретным файлом
Reset(файл)	Открывает файл для чтения
Rewrite(файл)	Открывает файл для записи
Append(файл)	Открывает файл для добавления данных в конец файла
Close(файл)	Закрывает файл
IOResult()	Возвращает ненулевое значение при наличии ошибки в работе с файлом
Rename(файл,имя)	Переименовывает файл
Erase(файл)	Удаляет файл

Текстовые файлы

Самыми простыми файлами являются текстовые. Они состоят из символов.

Подпрограмма	Описание
Read(файл, переменные)	Сохраняет прочитанные данные в переменной. Необходима минимум одна переменная
Write(файл, аргументы)	Записывает данные в файл. Необходим минимум один аргумент
Readln(файл, переменные)	Сохраняет прочитанные данные и переводит курсор на новую строку
Writeln(файл, аргументы)	Записывает данные в файл и переводит курсор на новую строку
EOF(файл)	Определяет, достигнут ли конец текстового файла
SeekEOF(файл)	Смещает позицию на конец текстового файла
SeekEOL(файл)	Истина если последний прочитанный символ маркер конца строки

Двоичные файлы

Типизированные и не типизированные файлы являются разновидностью двоичных файлов.

Подпрограмма	Описание
BlockRead(файл,переменная,размер)	Записывает данные из переменной в файл
BlockWrite(файл,переменная,размер)	Сохраняет прочитанный блок данных в переменной
FileSize(файл)	Возвращает размер файла
FilePos(файл)	Возвращает текущую позицию в файле
Seek(файл, позиция)	Меняет файловую позицию

Задание размера блока для не типизированных файлов

Размер блока передается в качестве необязательного дополнительного параметра при вызове подпрограммы Reset или Rewrite. Если размер блока не задан, то он равен 128 байт.

Режим открытия двоичных файлов

По умолчанию подпрограмма Reset открывает двоичные файлы для чтения и записи. Режим открытия файлов можно изменить, если присвоить предопределенной переменной FileMode нужный флаг. Режим должен быть задан до открытия файла.

Флаг	Режим
0	Чтение
1	Запись
2	Чтение и запись

Переопределение стандартного ввода-вывода

По умолчанию стандартным устройством ввода в консоли является клавиатура, а устройством вывода является дисплей. К этим устройствам привязаны предопределенные файловые переменные Input и Output. Переопределить стандартные устройства ввода-вывода можно привязав эти переменные к нужным файлам, а затем открыв файлы.

Глава 27. Каталоги

Помимо работы с файлами есть средства для работы с каталогами. Рассмотрим основные подпрограммы для работы с ними.

Подпрограмма	Описание
ChDir(каталог)	Смена текущего каталога
MkDir(каталог)	Создание каталога
RmDir(каталог)	Удаление пустого каталога

Глава 28. Структура программы

Программа на паскале начинается с заголовка.

Формат заголовка: program заголовок;

Далее идет подключение модулей. Затем идет описание пользовательских типов данных.

Потом идет объявление глобальных переменных и констант. Учтите, что нельзя делать объявление с использованием несуществующих типов.

Потом идет объявление меток и определения подпрограмм. Все эти элементы не являются обязательными и любой из них может отсутствовать.

В самом конце находится тело программы, которое является обязательным. Тело программы выглядит следующим образом:

begin
операторы
end.

Глава 29. Глобальные потоковые переменные

Глобальные потоковые переменные похожи на обычные глобальные переменные, но они видны во всех потоках, которые привязаны к программе. При объявлении глобальных потоковых переменных используется ключевое слово threadvar вместо var.

Часть 2. Сопровождение и повышение надежности программ

Глава 1. Комментарии

Комментарием называется поясняющий текст из одной или нескольких строк, который игнорируется компилятором. Поддерживается три типа комментариев.

Комментарий из нескольких строк старого стиля: (*текст*)
Комментарий из нескольких строк в стиле Turbo Pascal: {текст}
Комментарий из одной строки в стиле Delphi: //текст

Глава 2. Завершение программы

Нормальное завершение

Когда программа завершает свою работу она передает операционной системе код возврата, по которому можно узнать результат работы программы. Для того чтобы определить код возврата присвойте целочисленное значение глобальной предопределенной переменной ExitCode. Допустимы отрицательные числа.

Форсированное завершение

Для форсированного завершения работы предусмотрено три подпрограммы. Первая завершает работу в штатном режиме, а оставшиеся две подпрограммы завершают работу в аварийном режиме.

Процедура Halt завершает работу в штатном режиме и передает операционной системе код возврата, который берет в качестве аргумента.

Процедура RunError аварийно завершает работу. Эта процедура имеет две формы. Первая форма не берет аргументов и выводит сообщение с указанием адреса ошибки и передает системе нулевой код возврата. Вторая форма выводит сообщение с указанием адреса ошибки и передает системе код возврата, который процедура берет в качестве аргумента.

Процедура Error аварийно завершает работу и выводит сообщение, которое специфично для конкретного типа ошибки. Это процедура берет тип ошибки в качестве аргумента. Процедура Error была введена для совместимости с Delphi.

Ошибка	Описание
reNone	Нет ошибок
reOutOfMemory	Выход за границы памяти
reInvalidPtr	Неверный указатель
reDivByZero	Деление на ноль
reRangeError	Выход за границы диапазона
reIntOverflow	Целочисленное переполнение
reInvalidOp	Неверная операция
reZeroDivide	Деление на ноль
reOverflow	Переполнение
reUnderflow	Значение слишком маленькое
reInvalidCast	Ошибка преобразования типов
reAccessViolation	Ошибка доступа
rePrivInstruction	Ошибка в привилегированной инструкции
reControlBreak	Операция была прервана пользователем
reStackOverflow	Переполнение стека
reVarTypeCast	Ошибка преобразования вариантного типа
reVarInvalidOp	Ошибка операции с вариантным типом
reVarDispatch	Ошибка при работе с интерфейсом
reVarArrayCreate	Ошибка создания массива вариантного типа
reVarNotArray	Переменная вариантного типа не является массивом
reVarArrayBounds	Выход за границы вариантного массива
reAssertionFailed	Выброс ошибки был форсирован пользователем
reExternalException	Внешняя ошибка
reIntfCastError	Ошибка преобразования интерфейсов
reSafeCallError	Ошибка безопасного вызова
reQuit	Получен сигнал досрочного завершения

Адрес ошибки

Адрес выводимый в сообщениях подпрограмм аварийного завершения можно изменить при помощи предопределенной переменной ErrorProc.

Глава 3. Параметры командной строки

Часто бывает удобно предусмотреть возможность передачи программе параметров при запуске через командную строку. Если параметр, содержит пробел, то он берется в одинарные или двойные кавычки.

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. Нулевой параметр содержит имя исполняемого файла программы.

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;

Обработка и перехват исключений

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

Формат конструкции try...except:

Try
операторы
Except
On исключение do обработчик
end;

Если во время выполнения программы в блоке, идущем после ключевого слова Try, возникают ошибки, то выполнение кода вызвавшего ошибку приостанавливается и выполняется код обрабатывающий исключение. После выполнения обработчика состояние ошибки сбрасывается.

Формат конструкции try...finally:

Try
операторы
Finally
обработчик
end;

Здесь не нужно указывать тип исключения. При отсутствии исключения выполняется код из обеих секций. Если исключение произошло, то сразу выполняется код из секции Finally. Код из секции Try игнорируется. После выполнения обработчика состояние ошибки не сбрасывается.

Стандартные типы исключений

Стандартные типы исключений перечислены ниже. Для удобства разобьем исключения по группам. Подключите модуль SysUtils, чтобы иметь возможность использовать стандартные типы исключений. О подключении модулей будет рассказано в следующей главе.

Исключения, связанные с языком

Исключение	Описание
EAbort	Операция была прервана
EAbstractError	Вызов абстрактного метода
EArgumentException	Функции передан неверный аргумент
EArgumentOutOfRangeException	Аргумент функции выходит из диапазона
ENotImplemented	Функция не имеет реализации
ENoWideStringSupport	Широкие строки не поддерживаются
EInvalidCast	Ошибка преобразования типов
EPropReadOnly	Ошибка обращения к свойству
EPropWriteOnly	Ошибка обращения к свойству
EVariantError	Ошибка при работе с вариантным типом

Исключения, связанные с железом

Исключение	Описание
EBusError	Ошибка обращения к шине
EInvalidOp	Попытка выполнить неверную операцию

Исключения, связанные с памятью

Исключение	Описание
EAccessViolation	Ошибка доступа к области памяти
EHeapMemoryError	Ошибка выделения памяти из кучи
EInvalidPointer	Ошибка при работе с указателями
EOutOfMemory	Программа исчерпала доступный объем памяти
EStackOverflow	Переполнение стека

Математические исключения

Исключение	Описание
EConvertError	Ошибка преобразования типов
EDivByZero	Деление на ноль
EInterror	Ошибка при работе с целочисленной переменной
EIntfCastError	Ошибка преобразования целочисленных типов
EIntOverflow	Переполнение целочисленной переменной
EMathError	Ошибка в математической операции
ERangeError	Выход за пределы диапазона
EOverflow	Переполнение в переменной с плавающей точкой
EUnderflow	Ошибка округления дроби
EZeroDivide	Деление на ноль

Не поддерживаемые исключения

Исключение	Описание
EInvalidContainer	Игнорируется. Оставлено для совместимости с Delphi
EInvalidInsert	Игнорируется. Оставлено для совместимости с Delphi
EPackageError	Игнорируется. Оставлено для совместимости с Delphi
ESafecallException	Игнорируется. Оставлено для совместимости с Delphi

Исключения, связанные с операционной системой

Исключение	Описание
EControlC	Операция прервана нажатием Control+C
EExternal	Ошибка процессора или операционной системы
EExternalException	Программный выброс исключения
EInOutError	Ошибка в операции ввода-вывода
EOSError	Ошибка при обращении к операционной системе
EPrivilege	Программа пыталась выполнить недопустимую операцию
ENoThreadSupport	Ошибка при работе с потоком

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.

Встраивание модулей

При сборке программы каждый модуль компилируется отдельно. Затем модули встраиваются в исполняемый файл.

Стандартный модуль System

Модуль System имеет отличие от остальных модулей. Он автоматически подключается к программе без использования ключевого слова 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.

Зависимости

Подключение других модулей к вашему идет сразу после указания имени вашего модуля. Подключение других модулей не является обязательным. Рекурсивные зависимости недопустимы.

Альтернативный вариант подключения модулей

Обычно компилятор самостоятельно ищет модули. Можно так же использовать альтернативный вариант подключения с явным указанием имени файла.

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

Имя файла может быть полным, то есть содержать путь к файлу.

Написание модулей

Вы можете сами написать модуль. Структура модуля следующая:

unit имя;
interface
интерфейс
implementation
реализация
end.

Имена модулей

Имя модуля является обязательным. Оно должно совпадать с именем файла в котором находиться модуль.

Альтернативный формат имен модулей

Данная реализация Паскаля поддерживает так же альтернативный формат имен модулей. Имя модуля может состоять из двух частей, разделенных между собой точкой.

Интерфейсный раздел

Интерфейсный раздел содержит части кода, которые будут доступны после подключения модуля. Он состоит из трех секций. Вначале идет описания пользовательских типов данных и свойств. Затем идут объявления необходимых переменных и констант.

В конце находятся прототипы подпрограмм. Если какому-либо прототипу из интерфейсного раздела не соответствует определение из раздела реализации, то при компиляции программы возникнет ошибка.

Допустимо отсутствие прототипа при наличии определения в разделе реализации. Но в этом случае подпрограмма будет доступна только внутри модуля.

Раздел реализации

Этот раздел содержит две секции. Первая секция содержит переменные и константы из раздела, которые доступны только внутри модуля. Во второй секции находятся определения подпрограмм.

Замечания по поводу интерфейсного раздела и раздела реализации

Эти разделы могут быть пустыми и любая секция из этих разделов может отсутствовать если в ней нет необходимости. Обязательным является лишь наличие в модуле соответствующих ключевых слов.

Секции инициализации и завершения

После раздела реализации могут находиться необязательные секции инициализации и завершения.

Секция инициализации содержит код, выполняемый сразу после загрузки программы.

Код из секции завершения выполняется при завершении работы программы. Секция завершения идет сразу после секции инициализации.

Формат описания секции инициализации:

Initialization
операторы
end.

Ключевое слово Initialization может быть заменено ключевым словом begin.

Формат описания секции завершения:

Finalization
операторы
end.

Области видимости в модулях

Если их имена переменных или констант из интерфейсного раздела совпадают с другими именами, то они становятся недоступными. В этом случае используется следующая конструкция: модуль.имя

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.

Структурное программирование возникло как попытка исправить недостатки процедурного подхода. 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. Порождение одного объекта от другого называют наследованием.

Класс является основой для создания объекта. Фактически класс представляет собой еще один тип данных, которые определяются пользователем.

Глава 2. Классы

Спецификаторы доступа

Спецификаторы доступа позволяют ограничить доступ к элементам класса. Если спецификатор доступа не задан, то используется Public.

Спецификатор	Описание
Private	Класс доступен только в модуле, который его описывает
Strict Private	Члены класса доступны только внутри его методов
Protected	Члены класса доступны в производном классе даже, если тот описан в другом модуле
Public	Ограничений нет
Published	Публичные члены, которые доступны в инспекторе объектов. Массивы не могут быть членами класса

Описание класса

Синтаксис описания класса:

```
Type класс=class
элемент1
..
элементN
end;
```

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

Каждый элемент имеет следующий вид:

```
спецификатор
Объявления
прототипы
```

Вначале объявляются члены. Член объявляется как переменная. После объявления членов описываются прототипы методов. Прототип метода похож на определение подпрограммы, но у него отсутствует тело.

Статические поля

Данные-члены объявленные с модификатором static являются статическими. Они похожи на глобальные переменные доступные только внутри объекта.

Формат задания спецификаторов членов аналогичен формату задания спецификаторов методов.

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. Перед ключевым словом function или procedure идет идентификатор класса. Идентификатор и ключевое слово отделяются друг от друга точкой.

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(аргументы);
```

Обращение к текущему экземпляру объекта

Используйте ключевое слово self внутри тела своего метода, чтобы обратиться к текущему экземпляру объекта. Эта ключевое слово является псевдонимом текущего экземпляра.

Конструкторы и деструкторы

Каждый класс должен содержать специальные методы: конструктор и деструктор. Конструктор вызывается при создании объекта. Деструктор вызывается при уничтожении объекта. Имена этих методов произвольны. Конструктор может принимать аргументы на вход. Деструктор 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. Использование конструктора и деструктора обязательно.

Создание объекта

Перед тем как создать объект его необходимо объявить. Объявляется объект так же как переменная с тем отличием, что вместо типа указывается класс.

После объявления объекта необходимо вызвать конструктор.

Вызов конструктора и деструктора

Вызов конструктора приводит к выделению места под объект в динамической памяти. Освобождение памяти происходит при вызове деструктора.

Формат вызова конструктора: объект:=класс.конструктор(аргументы);
Деструктор вызывается как обычный метод.

Проверка объекта на принадлежность к классу

Чтобы проверить объект 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.

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(параметры); 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.

Методы обработки сообщений

Методы обработки сообщений применяются для обработки сообщений, которые поступают от операционной системы в программу.
Такие методы всегда являются виртуальными.

Прототипы методов обработки сообщений объявляются с модификатором Message. Определения методов так же содержат этот модификатор.

В определении вы можете указать идентификатор типа сообщения.

Он указывается после ключевого слова Message. Ключевое слово Message и идентификатор отделяются друг от друга пробелом. В конце конструкции ставится точка с запятой.

Идентификатор типа сообщений может представлять собой целочисленную константу или строку. Строковый идентификатор заключается в одинарные кавычки.

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. Свойства

Обычные свойства

Свойства позволяют ограничить доступ к членам. Обращение к свойству аналогично обращению к обычному члену.

Взаимодействие и передача данных между данными-членами и свойством происходит при помощи неявного обращения к члену или неявного вызова метода.

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 идентификатор:тип ограничитель;

Синтаксис задания ограничителя: спецификатор: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.

Свойства-массивы

Свойства-массивы позволяют организовать ограниченный доступ к массивам, которые находятся внутри объекта. При работе с таким свойством нужно указывать индекс элемента.

Обмен данными между массивом и свойством организуется через специальный метод, который содержит в списке параметров целочисленный параметр.

Объявление свойства: property свойство[параметр:тип] ограничитель;

Свойства по умолчанию

Свойства-массивы могут быть маркированы как свойства по умолчанию. Каждый класс может иметь только одно свойство по умолчанию. Свойства маркируются при помощи спецификатора default, который указывается сразу после ограничителя.

Свойства по умолчанию делают конструкции объект.свойство[индекс] и объект[индекс] эквивалентными.

Индексные свойства

Индексные свойства связаны с конкретными элементами массива.

Обмен данными между массивом и свойством организуется через специальный метод, который содержит в списке параметров целочисленный параметр с именем index.

Объявление свойства: property свойство тип index индекс ограничитель;

Свойства вне классов

Свойства могут быть использованы вне классов для ограничения доступа к глобальным переменным. Обмен данными осуществляется через функцию. Использование свойств вне классов доступно только в режиме совместимости с Object Pascal.

Глава 4. Наследование

Простое наследование

Класс может порождаться от другого класса, который называется родительским. Если класс имеет одного родителя, то наследование называют простым. При наследовании конструкция описывающая класс меняется. Сразу после ключевого слова class идет следующая конструкция: (родитель)

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.

Перегрузка методов в производном классе

Перегрузка методов аналогична перегрузке подпрограмм.

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

Отличия обычных и виртуальных методов

Виртуальные методы часто применяются при перегрузке 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 позднее 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.

Перегрузка виртуальных методов

Для того чтобы перегрузить виртуальный метод в производном классе используйте в прототипе метода модификатор override.

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

Вызов конструктора или деструктора родительского класса

Оператор inherited используется для вызова конструктора родительского класса. Он имеет две формы:

Первая форма: inherited;
Вторая форма: inherited конструктор(аргументы);

При использовании первой формы конструктору передаются те же аргументы что и методу дочернего класса. Во второй форме передаваемые аргументы задаются явно.

Оператор inherited можно также использовать для вызова деструктора.

Глава 5. Ссылки на классы

Что такое ссылка на класс?

В диалекте, реализованном в компиляторе Free Pascal, есть специальный тип данных, называемый ссылкой на класс. Переменные этого типа могут содержать указатели на описание класса.

Использование ссылок на классы в Free Pascal

Синтаксис описания ссылки на класс:

Type ссылка=class of 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.

Реализация интерфейсов

Интерфейс реализуется при помощи класса. Он должен идти сразу после описания интерфейса. В определении класса описание элементов должно совпадать с их описанием в интерфейсе. Для реализации интерфейса нужно два класса, которые используются как родительские классы. В качестве первого родителя используйте класс TInterfacedObject. Идентификатор интерфейса используйте как второго родителя.

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.
4. Спецификаторы, определяющие разновидности методов не используются.

Описание интерфейса

Интерфейс описывается аналогично классу, но вместо ключевого слова class используется ключевое слово interface.

Идентификатор интерфейса

Приведенная ниже информация специфична для Windows и не актуальна для других систем. В Windows интерфейс должен иметь 128 битовый идентификатор чтобы взаимодействовать с COM объектами. Такие идентификаторы обозначают термином GUID.

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

Определение GUID находится в разделе описаний модуля или программы.

Часть 4. Концепция объектов вне классов

Глава 1. 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 несколько отличий от представителей классов. Нельзя использовать свойства.
Память под объект без класса выделяется при входе в блок, который содержит объявление экземпляра объекта. Память освобождается при выходе из него.

Применение конструктора и деструктора необязательно и нужно только при наличии виртуальных методов или при использовании динамических объектов.

Спецификаторы доступа

Спецификаторов доступа в объектах без класса определяют область видимости экземпляра объекта.

Спецификатор	Описание
Private	Объект доступен только в своем модуле
Protected	Члены объекта доступны в производном объекте даже, если тот описан в другом модуле
Public	Ограничений нет

Описание объекта

Объект описывается аналогично классу, но вместо ключевого слова class используется ключевое слово object.

Определение методов

В определениях методов используется идентификатор объекта. В остальном определения методов объектов аналогичны определениям методов классов.

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. Экземпляр объекта объявляется как обычная переменная. Но вместо стандартного типа указывается 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.

Оператор with

Оператор with позволяет более удобно манипулировать экземпляром объекта и избежать излишнего использования его имени.

Синтаксис:
with экземпляр do
begin
операторы
end;

В качестве экземпляра 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 перегрузка и использование виртуальных методов. Для объектов без классов в прототипе перегружаемых 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 представителям класса.

Глава 2. Динамические объекты

Особенности динамических объектов

Использование обычных объектов часто влечет за собой большое потребление памяти. Поэтому Free Pascal предоставляет возможность ручного размещения объектов в динамической памяти.

Работа с объектами

Работа с экземпляром динамического объекта осуществляется через указатель.

Обращение к члену объекта: указатель^.член;
Вызов метода: указатель^.метод(аргументы);

Создание и удаление объектов

Создание экземпляра динамического объекта осуществляется подпрограммой New, а удаление экземпляра осуществляется подпрограммой Dispose.

Вызов конструктора и деструктора выполняется автоматически при вызове соответствующих подпрограмм.

Формат вызова подпрограммы New: New(указатель,конструктор(аргументы));
Формат вызова подпрограммы Dispose: Dispose(указатель,деструктор);

Альтернативный формат вызова подпрограммы New:
указатель:=New(объект,конструктор(аргументы));

Глава 3. Расширенные записи

Отличия расширенной записи от объекта без класса

Расширенные записи нужны для совместимости с Delphi 2006 или выше. Они доступны в режиме совместимости с Delphi.

Расширенные записи похожи на объекты без класса, но имеют ряд отличий.

Отличия расширенной записи от объекта без класса:

1. Не поддерживается наследование
2. Запрещен спецификатор published
3. Конструкторы и деструкторы не используются
4. Запрещены абстрактные и виртуальные методы
5. Запрещено использовать ключевое слово inherited

Описание расширенной записи

Синтаксис описания расширенной записи аналогичен описанию обычных записей, но в нем присутствуют спецификаторы доступа и прототипы методов. Правила определения методов аналогичны обычным объектам.

Часть 5. Помощники

Глава 1. Предназначение помощников

Данная реализация Паскаля поддерживает помощники. Помощник позволяет расширить функциональность существующего кода. Основным их достоинством является простота использования и возможность расширить существующий код без необходимости сильно его изменять. Фактически помощник является разновидностью декоратора. На данный момент помощники можно использовать вместе с классами, а так же простыми типами и записями.

Глава 2. Помощники классов

Ограничения

Помощник не является полноценным классом и имеет ряд жестких ограничений.

Ограничения помощника класса:

1. Запрещены конструкторы и деструкторы.
2. Невозможно определить члены и свойства
3. Запрещены абстрактные методы.
4. Виртуальные методы класса не могут быть переопределены. Модификатор overload скрывает их.
5. При перегрузке методов используется модификатор overload.

Создание помощника

Синтаксис описания помощника:

```
Type помощник=class helper for класс;
прототипы
End;
```

В определениях методов используется идентификатор помощника.

Вызов методов

Вызов методов помощника осуществляется через объект привязанного к помощнику класса.

Глава 3. Помощники записей

Помощники записей имеют специфические ограничения.

Ограничения помощников записей:

1. Могут использоваться только вместе с записями
2. Конструкторы и деструкторы запрещены
3. Доступны только в режиме совместимости с Object Pascal или Delphi
4. Оператор inherited можно использовать только в режиме совместимости с Object Pascal
5. Помощник расширенной записи может обращаться только к публичным членам

Синтаксис описания помощника записи аналогичен синтаксису помощника класса. Вызов методов помощника осуществляется через экземпляр структуры.

Глава 4. Помощники простых типов

К простым типам данных относятся типы данных, которые были рассмотрены в третьей главе первой части этой книги. Помощники простых типов могут работать только с этими типами.

Помощники простых типов доступны только в режиме совместимости с Delphi или Object Pascal. Для их использования так же следует задействовать директиву {$modeswitch typehelpers}.

В остальном они ведут себя как обычные помощники.

Часть 6. Взаимодействие с компилятором

Глава 1. Инструкции для компилятора

Директивами называют специальные инструкции управляющие процессом компиляции. Директивы вставляются прямо в исходный код программы.

Общий вид директивы: {$директива}

Если директиве нужно передать параметры, то синтаксис меняется и принимает следующий вид: {$директива параметры}

Глава 2. Основные директивы

Необходимый минимум

В этой главе представлены основные директивы. Полную информацию и список всех директив можно найти в официальной документации по Free Pascal.

Глобальные директивы

Глобальные директивы влияют на весь процесс компиляции. Они должны располагаться вначале файла с исходным кодом.

Директива	Параметры	Описание
DEBUGINFO	ON или OFF	Определяет наличие отладочной информации
SMARTLINK	ON или OFF	Включает или отключает умную компоновку
POINTERMATH	ON или OFF	Размещает или запрещает указатели в выражениях
CODEPAGE	Кодировка	Явно задает кодировку файла с исходным кодом
MINSTACKSIZE	Размер в байтах	Задает минимальный размер стека
MAXSTACKSIZE	Размер в байтах	Задает максимальный размер стека
MEMORY	Стек, куча	Задает размер стека и кучи
O	Нет	Включает оптимизацию второго уровня
S+	Нет	Включает проверку стека
S-	Нет	Отключает проверку стека
MODE	Флаг режима	Устанавливает режим совместимости
PROFILE	ON или OFF	Определяет наличие информации для профайлера
EXTENDEDSYNTAX	ON или OFF	Размещает или запрещает не возвращать значение из функции

Локальные директивы

Локальные директивы влияют только на текущий обрабатываемый файл и могут располагаться в любом месте.

Директива	Параметры	Описание
OPTIMIZATION	Флаг оптимизации	Задает режим оптимизации
PACKETSET	Размер в байтах	Определяет количество памяти выделяемой для хранения множеств
ALIGN	Граница выравнивания в байтах	Задает выравнивание данных
PACKETRECORDS	Граница выравнивания в байтах	Задает выравнивание данных в записях
CALLING	Флаг формата	Определяет формат передачи параметров, используемый по умолчанию
BITPAKING	ON или OFF	Включает или выключает автоматическое упаковывание записей
COOPERATOR	ON или OFF	Разрешает или запрещает комбинированные операторы
GOTO	ON или OFF	Разрешает или запрещает оператор goto
INLINE	ON или OFF	Разрешает или запрещает модификатор inline
STATIC	ON или OFF	Разрешает или запрещает ключевое слово static
MACRO	ON или OFF	Разрешает или запрещает макросы
INCLUDE	Имя файла	Подключает файл с исходным кодом
I	Имя файла	Подключает файл с исходным кодом
H+	Нет	Делает эквивалентными типы String и AnsiString
H-	Нет	Делает эквивалентными типы String и ShortString
I+	Нет	Включает выброс исключений при ошибках в работе с файлами
I-	Нет	Выключает выброс исключений при ошибках в работе с файлами
WAIT	Нет	Приостанавливает компиляцию и ждет нажатия клавиши
HINT	Текст	Выводит на экран подсказку
WARNING	Текст	Выводит на экран предупреждение

Глава 3. Игнорируемые директивы

Ряд директив компилятор игнорирует. Они нужны только для обеспечения совместимости с другими реализациями языка Паскаль.

Директива	Тип	Обеспечивает совместимость с реализацией
EXTENDEDSYM	Локальная	Delphi
EXTERNALSYM	Локальная	Delphi
HPPEMT	Локальная	Delphi
NODEFINE	Локальная	Delphi
STRINGCHECKS	Локальная	Delphi
LIBEXPORT	Локальная	Objective Pascal
WEAKPACKAGEUNIT	Глобальная	Delphi
N	Глобальная	Turbo Pascal

Глава 4. Условные директивы

Особые директивы

Особой разновидностью локальных директив являются условные директивы. Они позволяют организовать выборочную обработку других директив в зависимости от наличия или отсутствия специальных идентификаторов.

Управление идентификаторами

Для работы с идентификаторами предусмотрены две синтаксические конструкции.

Объявление идентификатора: {$DEFINE идентификатор}
Отмена объявления идентификатора: {$UNDEFINE идентификатор}

Знакомство с условными директивами

Рассмотрим основные условные директивы.

Директива IFDEF обрабатывает блок, если идентификатор объявлен. Директива ELSE и следующий за ней блок не являются обязательными.

Синтаксис:
{$IFDEF идентификатор}
блок директив или кода на Паскале обрабатываемый, когда идентификатор объявлен
{$ELSE}
блок директив или кода на Паскале обрабатываемый, когда идентификатор отсутствует
{$ENDIF}

Директива IFUNDEF обрабатывает блок, если идентификатор отсутствует. Директива ELSE и следующий за ней блок не являются обязательными.

Синтаксис:
{$IFUNDEF идентификатор}
блок директив или кода на Паскале обрабатываемый, когда идентификатор отсутствует
{$ELSE}
блок директив или кода на Паскале обрабатываемый, когда идентификатор объявлен
{$ENDIF}

Глава 5. Подключение файлов

Вы можете подключить один файл с исходным кодом к другому при помощи локальной директивы INCLUDE или I. Обе директивы берут имя файла в качестве параметра.

Если имя файла содержит пробелы, оно заключается в одинарные кавычки. Если расширение файла не указано в его имени, то автоматически добавляется расширение .pp.

Связанные между собой файлы рассматриваются при компиляции как одно целое.

Глава 6. Макросы

Макрос представляет собой идентификатор, который заменяет выражение. Этот идентификатор в дальнейшем может использоваться вместо выражения.

Синтаксис: {$define идентификатор:=выражение;}

Рекурсивное определение макросов недопустимо. Макросы, определенные в интерфейсной части доступны только внутри модуля.

Глава 7. Формат передачи аргументов по умолчанию

Используемый по умолчанию формат передачи аргументов в подпрограммы можно изменить с помощью глобальной директивы CALLING. Доступные форматы даны в таблице ниже.

Формат	Описание
CDECL	Формат языка Си
CPPDECL	Формат языка Си++
FPCCALL	Старый формат Free Pascal
INLINE	Использование встраиваемых подпрограмм если возможно
PASCAL	Формат языка Паскаль
REGISTER	Передача аргументов через регистры процессора
SAFECALL	Аргументы помещаются в стек справа налево. Очистку стека выполняет вызванная подпрограмма. Состояние регистров сохраняется перед вызовом и восстанавливается после выхода.
STDCALL	Аргументы помещаются в стек справа налево. Очистку стека выполняет вызванная подпрограмма.
SOFTFLOAT	Эмуляция математического сопроцессора на процессорах ARM

Глава 8. Типы программ

Free Pascal позволяет создавать консольные программы и программы с графическим интерфейсом. Тип программы определяется глобальной директивой APPTYPE. Флаг CONSOLE соответствует консольной программе, а флаг GUI программе с графическим интерфейсом.

Переключение между режимами

Переключение между режимами осуществляется с помощью директивы MODE и флагов. Список флагов дан в таблице ниже.

Флаг	Режим
FPC	Режим Free Pascal
TP	Режим совместимости с Turbo Pascal
Delphi	Режим совместимости с Delphi
DelphiUnicode	Режим совместимости с современными версиями Delphi
OBJFPC	Режим совместимости с Object Pascal
MAC	Режим совместимости с Mac Pascal

Особенности режимов

Рассмотрим особенности каждого режима.

Особенности режима Free Pascal:

1. Вы должны использовать оператор взятия адреса для присваивания значений переменным процедурного типа.
2. При использовании предварительного объявления подпрограмм список параметров в объявлении должен быть идентичен списку в определении.
3. Разрешена перегрузка подпрограмм.
4. Разрешены вложенные комментарии.
5. Не загружается модуль ObjPas.
6. Можно использовать ключевое слово cvar.
7. Переменные типа PChar автоматически преобразуются в строки.
8. Тип String по умолчанию эквивалентен типу ShortString.

Особенности режима совместимости с Delphi:

1. Вы не можете использовать оператор взятия адреса для присваивания значений переменным процедурного типа.
2. При использовании предварительного объявления подпрограмм список параметров в объявлении должен быть идентичен списку в определении.
3. Запрещена перегрузка подпрограмм.
4. Запрещены вложенные комментарии.
5. Модуль ObjPas загружается автоматически.
6. Тип Integer эквивалентен типу LongInt.
7. Тип String по умолчанию эквивалентен типу AnsiString.

Особенность режима совместимости с современными версиями Delphi

Этот режим аналогичен обычному режиму совместимости с Delphi, но рассчитан на совместимость с современными версиями Delphi – Delphi 2009 и выше. Главным отличием этого режима от обычного является использование юникода для символов и строк. На данный момент совместимость с современными версиями Delphi находиться на начальном уровне.

Особенности режима совместимости с Turbo Pascal:

1. Множества занимают в памяти один байт, если в них меньше 257 элементов.
2. Вы не можете использовать оператор взятия адреса для присваивания значений переменным процедурного типа.
3. При использовании предварительного объявления подпрограмм список параметров в объявлении должен быть идентичен списку в определении.
4. Запрещена перегрузка подпрограмм.
5. Запрещены вложенные комментарии.
6. Модуль ObjPas не загружается.
7. Нельзя использовать ключевое слово cvar.
8. Тип String по умолчанию эквивалентен типу ShortString.

Часть 7. Введение в библиотеку Lazarus Component Library

Глава 1. 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 взаимодействие с программой осуществляется через графические элементы. 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 представляет собой белый квадрат и нужно для ввода информации.

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 меню. 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.

Глава 2. Особенности библиотеки

В терминах библиотеки Lazarus Component Library элемент графического интерфейса называют компонентом. Каждый компонент реализуется как объект соответствующего класса, который наследуется от некоторого базового класса.

Сама работа программы определяется совокупностью обработчиков событий. Событием называется действие пользователя или операционной системы. В качестве примера события можно назвать щелчок левой кнопки мыши по элементу интерфейса программы. Обработчик события пишется в теле метода класса.
Организация библиотеки Lazarus Component Library идентична организации библиотеки Visual Component Library, используемой в Delphi и C++ Builder.

Глава 3. Основные элементы интерфейса

Окно

Каждая программа с графическим интерфейсом содержит как минимум одно окно, внутри которого отображаются остальные элементы интерфейса. Часто окно имеет заголовок. Окно является объектом класса TForm.

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

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
OnActive	Происходит при активации окна
OnDeactive	Происходит при деактивации окна

Кнопка

Кнопка является объектом класса TButton.

События

Событие	Описание
OnClick	Щелчок на кнопке
OnFocus	Получение фокуса

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

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 находится 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
WordWrap	Логическое значение	Перенос текста по словам
WantTabs	Логическое значение	Разрешает или запрещает ввод символов табуляции
MaxLength	Целое число	Максимальная длина одной строки
Lines	Объект типа TStrings	Дает доступ к строкам текста
Font	Объект типа TFont	Шрифт 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
Stretch	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.

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

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

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.

Глава 4. 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

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 5. 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 программу
Suspend	Нет	Целое число	Приостанавливает программу
Resume	Нет	Целое число	Возобновляет работу программы

Константы приоритета

Константа	Описание
ppHigh	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

Часть 8. Среда Lazarus

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, содержится редактор исходного кода и отладчик. Редактор исходного кода представляет собой текстовый редактор с подсветкой синтаксиса.

В основе быстрых средств разработки лежит идея проектирования пользовательского интерфейса в редакторе форм. Среда Lazarus совмещает обе концепции.

К практическим достоинствам Lazarus относится 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, так же поддерживает несколько платформ.

Глава 2. Проекты

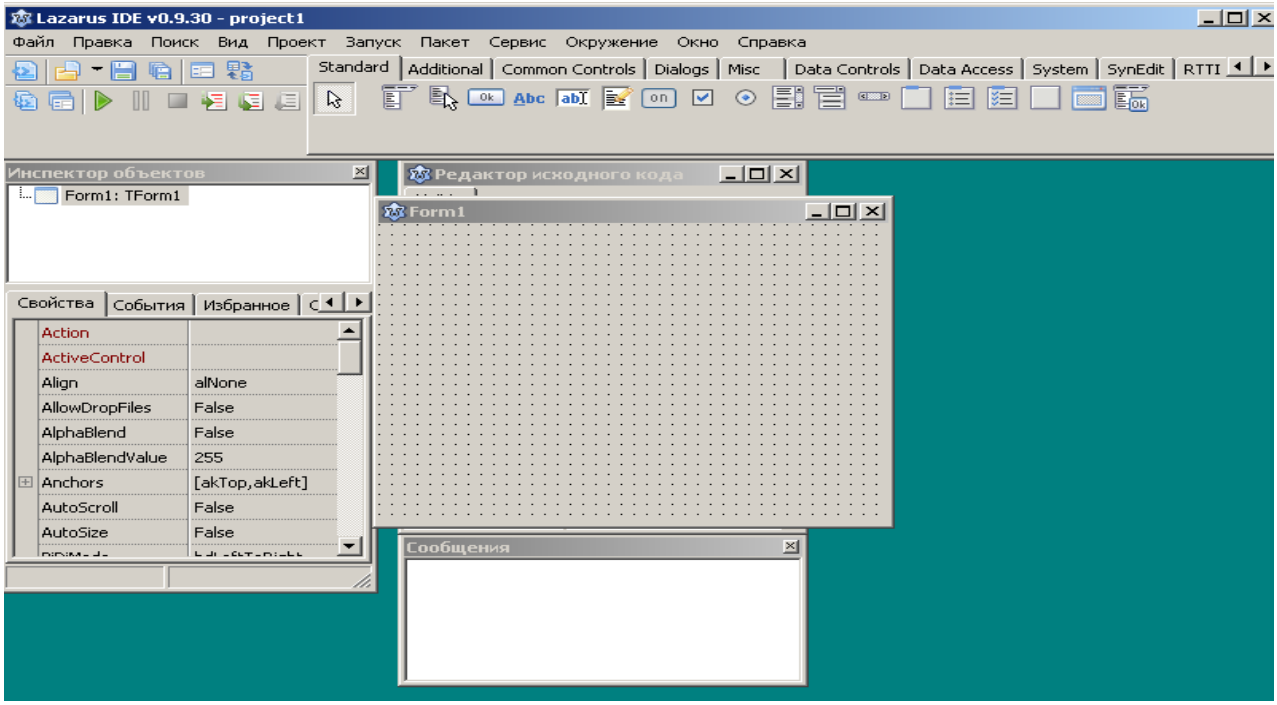
Проектом называется совокупность файлов, которые используются для сборки программы. Список этих файлов дан в таблице ниже.

Расширение файла	Описание
pas	Исходный код на паскале
lpr	Исходный код главного модуля
lfm	Описание формы
lpi	Описание проекта

Содержимое главного модуля и файлов описаний генерируется автоматически.

При запуске Lazarus автоматически создает новый проект.

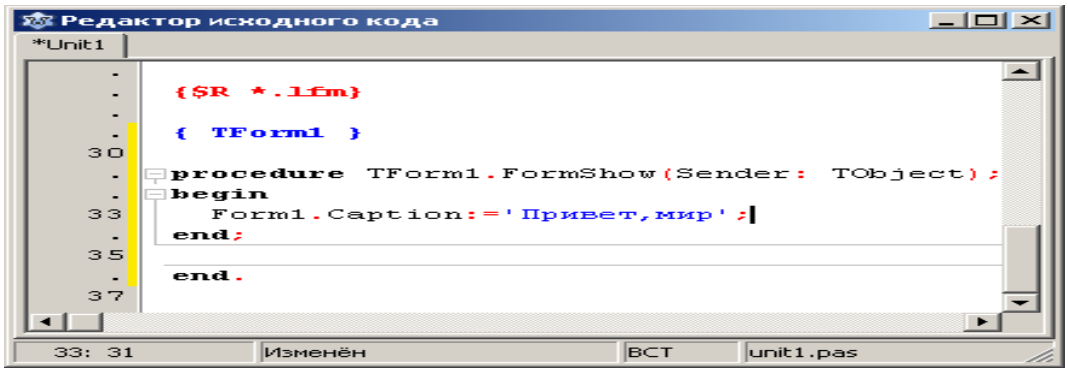
Вы увидите следующие окно:



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

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. Пример обработчика события дан на рисунке ниже.



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 диалекта реализованного во 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 которые способны 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 Основы программирования в среде Lazarus - http://mansurov-oshtu.ucoz.ru/

4 Поляков Андрей Валерьевич. Руководство программиста Free Pascal 2.4.2 - http://av-mag.ru/doc/fpc-programmer-manual.htm

5 Свободное программное обеспечение. FREE PASCAL для студентов и школьников / Ю. Л. Кетков, А. Ю. Кетков. — СПб.: БХВ-Петербург, 2011

6 Free Pascal и Lazarus: Учебник по программирова-нию / Е.Р.Алексеев,О.В.Чеснокова, Т.В.Кучер — М. : ALT Linux ; Издательский дом ДМК-пресс, 2010

7 Алексеев Е.Р., Чеснокова О.В., Кучер Т.В. Самоучитель по программированию на Free Pascal и Lazarus. - Донецк.:ДонНТУ, Технопарк ДонНТУ УНИТЕХ, 2011

8 Глушаков С. В., Коваль А. В., Смирнов С. В.Г. Язык программирования С++: Учебный курс, ООО «Издательство АСТ» 2001

9 В.В.Фаронов. ОСНОВЫ ТУРБО ПАСКАЛЯ, СП УИЦ «МВТУ-ФЕСТО ДЕДАКТИК», Москва 1991

10 Borland Delphi 6. Руководство разработчика. : Пер. с англ. — М. : Издательский дом “Вильямс”, 2002