

Е.А.Попов

Экспресс курс программирования в Lazarus

Редакция 105

2011 — 2021 год

Содержание

Введение.....	4
Условные обозначения.....	4
Условия распространения книги и обратная связь с автором.....	4
Часть 1. Основные сведения о диалекте Free Pascal.....	5
Глава 1. Идентификаторы и ключевые слова.....	5
Глава 2. Конфликты идентификаторов.....	6
Глава 3. Хранение данных.....	6
Глава 4. Области видимости данных.....	9
Глава 5. Основные операторы.....	10
Глава 6. Условные операторы.....	13
Глава 7. Циклы.....	14
Глава 8. Подпрограммы.....	16
Глава 9. Математика.....	19
Глава 10. Манипуляции с байтами.....	20
Глава 11. Модификаторы подпрограмм.....	20
Глава 12. Указательные подсказки.....	23
Глава 13. Приращение и уменьшение.....	24
Глава 14. Консоль.....	24
Глава 15. Массивы.....	25
Глава 16. Динамические массивы.....	27
Глава 17. Указатели.....	28
Глава 18. Динамическое распределение памяти.....	30
Глава 19. Процедурный тип.....	31
Глава 20. Множества.....	32
Глава 21. Обработка строк.....	34
Глава 22. Дополнительные подпрограммы для работы со строками и динамическими массивами.....	37
Глава 23. Записи.....	37
Глава 24. Значения по умолчанию.....	39
Глава 25. Упакованные сущности.....	39
Глава 26. Перегрузка операторов.....	40
Глава 27. Определение типа во время выполнения программы.....	41
Глава 28. Файлы.....	41
Глава 29. Каталоги.....	44
Глава 30. Структура программы.....	44
Глава 31. Ограничение доступа к глобальным переменным.....	44
Глава 32. Поточковые переменные.....	45
Часть 2. Сопровождение и повышение надежности программ.....	45
Глава 1. Комментарии.....	45
Глава 2. Утверждения.....	45
Глава 3. Параметры командной строки.....	46
Глава 4. Завершение программы.....	46
Глава 5. Обработка ошибок.....	48
Глава 6. Модули.....	51
Глава 7. Динамические библиотеки.....	54
Часть 3. Объектно-ориентированное программирование.....	57
Глава 1. Принципы объектно-ориентированного программирования.....	57
Глава 2. Классы.....	58
Глава 3. Свойства.....	63
Глава 4. Наследование.....	64
Глава 5. Интерфейсы.....	66

Часть 4. Концепция объектов вне классов.....	67
Глава 1. Объекты без классов.....	67
Глава 2. Динамические объекты.....	69
Глава 3. Расширенные записи.....	70
Часть 5. Обобщения.....	71
Глава 1. Шаблоны.....	71
Глава 2. Обобщенные расширенные записи.....	72
Часть 6. Помощники.....	72
Глава 1. Предназначение помощников.....	72
Глава 2. Помощники классов.....	72
Глава 3. Помощники записей.....	73
Глава 4. Помощники простых типов.....	73
Часть 7. Взаимодействие с компилятором.....	74
Глава 1. Инструкции для компилятора.....	74
Глава 2. Игнорируемые директивы.....	74
Глава 3. Подключение файлов.....	74
Глава 4. Макросы.....	75
Глава 5. Типы программ.....	75
Глава 6. Условные директивы.....	76
Глава 7. Основные директивы.....	77
Глава 8. Избирательное задействование языковых возможностей.....	79
Глава 9. Режимы совместимости.....	80
Часть 8. Введение в библиотеку Lazarus Component Library.....	82
Глава 1. Описание элементов графического интерфейса.....	82
Глава 2. Особенности библиотеки.....	84
Глава 3. Основные элементы интерфейса.....	85
Глава 4. Диалоги.....	91
Глава 6. Таймер.....	92
Глава 5. Запуск программ.....	93
Часть 9. Среда Lazarus.....	94
Глава 1. Средства быстрой разработки.....	94
Глава 2. Проекты.....	94
Глава 3. Проектирование в Lazarus.....	95
Заключение.....	96
Список литературы.....	97

Введение

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

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

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.

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

Автор этого справочника является независимым разработчиком программного обеспечения с многолетней практикой программирования. Данная книга возникла как результат желания помочь другим программистам. Надеюсь, она окажется полезной читателю.

Излагаемый материал рассчитан на Free Pascal версии 3.2 и среду Lazarus версии 2.0.10. Однако изложенные сведения применимы и к более новым версиям.

Условные обозначения

Книга разбита на части. Каждая часть состоит из нескольких глав.

Главы могут включать в себя несколько разделов. Заголовок раздела главы выделяется подчеркиванием.

Синтаксические конструкции выделяются *курсивом*.

Условия распространения книги и обратная связь с автором

Разрешается свободное распространение не модифицированной копии электронного варианта книги в некоммерческих целях. Иное использование разрешено только с согласия автора. Связаться с ним можно посылв 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, overload, 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?

Компьютерные программы полезны тем, что могут обрабатывать различные данные. Переменные используют для хранения этих данных. Каждая переменная имеет имя по которому к ней можно обращаться. Так же каждая переменная имеет свой тип, который определяет вид хранимых данных. Размер конкретной переменной зависит от ее типа и от системы 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.

Допускается использование встроенных или самостоятельно определяемых типов данных.

Привязка данных

Двум переменным можно назначить одну и ту же область памяти.

Синтаксис: *var имя: тип absolute цель;*

Целевая переменная должна быть объявлена заранее.

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

Логические типы

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

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

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 до 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

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

Совместно с логическими переменными можно использовать идентификаторы *True* и *False*. Идентификатор *True* соответствует истине. Лжи соответствует идентификатор *False*.

Перечисления

Множество однотипных констант удобно преобразовать в самостоятельно определяемых типов данных, называемый перечислением.

Синтаксис:

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

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: *Типе псевдоним*=тип;

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

Интервальный тип

Интервальный тип данных является еще одним видом самостоятельно определяемого типа данных. Он создается при помощи явного задания интервала допустимых значений.

Делается это при помощи следующей конструкции: *type тип*=начало..конец;

Если переменная интервального типа превысит заданный диапазон, то компилятор выдаст соответствующее сообщение во время компиляции.

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 оператором.

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`;

Особенности математических операторов

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

Оператор возведения в степень работает только с целыми положительными числами.

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

Комбинированные операторы

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

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 операторами. 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

Условия в программах

Условные операторы позволяют организовать ветвление в программе. Существуют два условных оператора. Рассмотрим их подробнее.

Оператор ветвления

Оператор ветвления имеет две формы. Первая форма наиболее проста. Начнем с нее.

Синтаксис первой формы: *if условие then оператор;*

Если условие истинное, то выполняется оператор идущий после ключевого слова *then*. В более сложных случаях можно использовать вторую форму.

Синтаксис второй формы: *if условие then оператор1 else оператор2;*

Если условие истинно, то выполняется оператор1. Иначе выполняется оператор2.

Прежде чем идти дальше стоит сказать еще несколько слов об этом операторе.

Вместе с оператором *if* можно использовать блочный оператор. Но есть одна особенность, которая касается второй формы оператора *if*.

Если после ключевого слова *then* и после ключевого слова *else* идет блочный оператор, то меняется синтаксис второго блочного оператора. В этом блочном операторе должна отсутствовать точка с запятой после ключевого слова *end*.

Оператор множественного выбора

Оператор множественного выбора позволяет выполнить разные действия в зависимости от значения выражения.

Синтаксис:

```
case выражение of  
значение1:оператор1;  
...  
значениеN:операторN;  
else  
оператор;  
end;
```

Если значение выражения не совпадает ни с одним из перечисленных значений, то выполняется оператор, следующий за ключевым словом *else*. Ключевое слово *else* и следующий за ним оператор не являются обязательными операторами.

Глава 7. Циклы

Что такое цикл?

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

Цикл со счетчиком

Цикл со счетчиком выполняет оператор заданное число раз. Синтаксис этого оператора имеет две формы.

Первая форма: *for счетчик:=начало to конец do оператор;*

Вначале определяется начальное и конечное значение счетчика. Счетчик увеличивается на единицу в конце каждой итерации цикла.

Вторая форма: *for счетчик:=начало downto конец do оператор;*

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

Проход по элементам множества или массива

Во Free Pascal 2.4.2 появился новый вид цикла, который проходит по всем элементам множества или массива.

Синтаксис: *for переменная in множество do оператор;*

На каждой итерации переменная получает значение текущего элемента. Этот цикл также может применяться к строкам.

Цикл с предусловием

Цикл с предусловием выполняет оператор пока условие истинно. Проверка условия происходит перед выполнением оператора.

Синтаксис: *while условие do оператор;*

Цикл с постусловием

Цикл с постусловием выполняет оператор пока условие истинно. 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;
```

В этом цикле 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(параметры);

Объявления

begin

тело

end;

Параметры в списке параметров отделяются друг от друга точкой с запятой.

Определение функции

Функция определяется следующим образом:

```
function функция(параметры):тип;  
Объявления  
begin  
тело  
end;
```

Напишите следующую конструкцию в конце тела функции, чтобы вернуть 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 процедуры 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

Начиная с Free Pascal 3.2 появилась возможность не указывать тип параметра. В этом случае при вызове подпрограммы в соответствующий параметр будет просто передан адрес аргумента.

Не у всех параметров может отсутствовать тип. Отсутствие типа допустимо для параметров-переменных, а так же для постоянных и выходных параметров.

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. Перегружаемые подпрограммы должны различаться количеством параметров.

Использование типизированных констант в подпрограммах

Типизированные константы, объявленные внутри подпрограмм, ведут себя почти так же как обычные переменные и могут менять свое значение в процессе выполнения подпрограммы. Основное отличие типизированных констант от переменных заключается в том что они не уничтожаются при выходе из подпрограммы и сохраняют свое значение между вызовами подпрограммы.

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
Ln(число)	Вычисляет натуральный логарифм положительного числа

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. Манипуляции с байтами

Так же в распоряжении программиста имеются подпрограммы для манипуляции с байтами внутри переменных. Список этих подпрограмм дан в таблице ниже.

Подпрограмма	Описание
Hi(переменная)	Возвращает старший байт/слово из переменной
Lo(переменная)	Возвращает младший байт/слово из переменной
Swap(переменная)	Меняет местами младший и старший байт/слово в переменной

Глава 11. Модификаторы подпрограмм

Общие сведения

Для использования модификатора необходимо изменить определение или прототип подпрограммы. Точка с запятой в конце имени подпрограммы дополняется следующей конструкцией: *модификатор*;

Если модификатору требуется передать параметр, то конструкция принимает следующий вид: *модификатор параметр*;

Модификаторы должны следовать друг за другом если необходимо использовать несколько модификаторов одновременно.

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

Перегружаемые подпрограммы могут маркироваться модификатором *overload*.

Использование этого модификатора не является обязательным и нужен только если подпрограммы расположены в другом модуле.

Псевдонимы подпрограммы

Подпрограмма может иметь дополнительное имя, которое можно использовать для ее вызова. Это имя называют псевдонимом. Псевдоним задается модификатором *alias*.

Формат использования модификатора: *alias*: 'псевдоним'

Вместо модификатора *alias* можно использовать модификатор *public*.

Формат использования модификатора: *public name* 'псевдоним'

Начиная с третьей версии компилятора ключевое слово *alias* является устаревшим. Оно оставлено для совместимости со старым кодом.

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

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 тела.

Внешние подпрограммы

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

Можно явно указать файл в котором содержится подпрограмма при помощи второй формы модификатор *external*.

Формат использования второй модификатора: *external*: 'файл'

Подпрограммы с переменным числом аргументов

Подпрограммы загружаемые из библиотек написанных на языке Си или Си++ могут иметь переменное число аргументов. Прототип таких подпрограмм должны быть маркированы модификатором *varargs*.

Вместе с этим модификатором обязательно следует использовать модификатор *cdecl*.

Обработчики прерываний

Подпрограммы, которые являются обработчиками прерываний маркируются модификатором *interrupt*. Обработчики прерываний не поддерживаются на некоторых платформах.

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.

Возврат к основной программе

После выполнения своей работы подпрограмма возвращает управление основной программе. Модификатор *noreturn* используется, чтобы запретить возврат к основной программе.

Управление стеком

По умолчанию при выполнении своей работы подпрограммы задействуют стек. Отключить использование стека можно при помощи модификатора *nostackframe*.

Этот модификатор следует использовать с осторожностью так как большинству подпрограмм нужен стек.

Локальные подпрограммы

Локальные подпрограммы доступны только в своих модулях и не видны в других частях программы. Локальные подпрограммы не имеют прототипов в интерфейсной части модуля. Подробнее о модулях будет рассказано во второй части книги.

Локальные подпрограммы могут быть маркированы модификатором *local*. Этот модификатор не является обязательным и введен для совместимости с Kylix.

Публичные подпрограммы

Публичные подпрограммы похожи на локальные, но имеют одно отличие. Публичная подпрограмма доступна для вызова из разных частей программы если подпрограмма находится во внешнем объектном файле.

Публичные подпрограммы маркируются модификатором *public*.

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	Компилятор будет пытаться передавать аргументы через регистры процессора.
saveregisters	Состояние регистров сохраняется перед вызовом подпрограммы.

Глава 12. 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.

Чтобы маркировать константы и переменные замените точку с запятой в конце объявления на пробел с последующей указательной директивой. 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 13. Приращение и уменьшение

Подпрограмма Inc увеличивает значение целочисленной переменной на указанную величину. Подпрограмма Dec выполняет обратное действие.

Обе подпрограммы берут имя переменной и значение величины в качестве аргументов. Вторым аргументом является необязательным и может пропущен. В этом случае переменная будет уменьшена или увеличена на единицу.

Глава 14. 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 друг от друга 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.

Для ограничения числа выводимых символов в целом числе или строке используйте одиночный спецификатор.

Глава 15. 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

Для инициализации массива 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 длины в качестве аргументов подпрограмм применяются специальные параметры, называемые открытыми массивами.

Формат объявления параметра: *имя: array of min*

Псевдонимы массивов

Массивы относятся к структурированным типам данных. Для них так же как для обычных типов можно определить псевдоним. Для этого предусмотрено две конструкции.

Вариант для обычного массива: *type псевдоним=array[размерность] of min;*

Вариант для динамического массива: *type псевдоним=array of min;*

Эти псевдонимы могут быть использованы позже при объявлении соответствующих переменных.

Выход за границы

Выход за границы массива может привести к повреждению данных. Также он делает вашу программу неустойчивой и позволяет внедрить в нее посторонний код.

Определение границ

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 16. Динамические массивы

Особенности динамических массивов

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

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

Объявление массива

Синтаксис объявления динамического массива отличается от объявления обычного массива. Он имеет две формы.

Пример объявления одномерного динамического массива:

```
var simple: array of Byte;
```

Пример объявления двумерного динамического массива:

```
var matrix: array of array of Byte;
```

Нумерация в динамических массивах

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

Выделение памяти

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.

Конструктор динамического массива

Начиная с третьей версии Free Pascal были добавлены конструкторы массивов. Конструктор выделяет память под массив и заполняет его данными.

Компилятор автоматически создает для каждого динамического массива свой конструктор. Он имеет имя `Create`.

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

Для многомерных массивов необходим рекурсивный вызов конструктора. Пример такого вызова вы можете увидеть в соответствующем разделе официальной документации.

Глава 17. 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

Поставьте знак `^` после имени указателя, чтобы выполнить операцию разыменования.

Нулевые указатели

Нулевым называется указатель, который не содержит адреса. 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 18. 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.

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.

Выделение 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.

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
<code>MemSize(блок)</code>	Возвращает размер блока памяти
<code>SizeOf(элемент)</code>	Возвращает размер элемента
<code>addr(аргумент)</code>	Возвращает адрес аргумента

Глава 19. Процедурный тип

Указатели на подпрограммы

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

Определение для процедуры: `type tnp=procedure(параметры);`

Определение для функции: `type tnf=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 20. 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.

Синтаксис: *var имя: set of тип;*

В качестве допустимых типов можно использовать тип Byte или тип Char.

Заполнение множеств

Для заполнения множества элементами воспользуйтесь следующей конструкцией: *множество:=[список];*

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

Ограничения множеств

Пустым называют множество, которое не содержит элементов. 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 бинарных операторов.

Оператор	Описание
+	Объединение
*	Пересечение
-	Разность
\oplus	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 21. 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

Внутреннее 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 взаимодействия 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.

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.

Получение строкового представления числа в различных форматах

Получение строкового представления числа в различных форматах часто бывает полезно. Для этой цели предусмотрено несколько функций.

Подпрограмма	Описание
OctStr(число)	Преобразует число в строку с восьмеричным представлением числа
BinStr(число)	Преобразует число в строку с двоичным представлением числа
HexStr(число)	Преобразует число в строку с шестнадцатеричным представлением числа

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
UpperCase(строка)	Преобразует строку в верхний регистр
Copy(строка,позиция,размер)	Возвращает часть строки, то есть подстроку
Pos(подстрока,строка)	Возвращает позицию нахождения подстроки

Глава 22. Дополнительные подпрограммы для работы со строками и динамическими массивами

В этой главе рассмотрим три подпрограммы, которые заслуживают отдельного внимания.

Функция `Concat` берет несколько строк в качестве аргументов и возвращает результат их слияния. Максимальный размер результирующей строки составляет 255 символов потому при необходимости она обрезается.

Поведение двух последних подпрограмм зависит от версии Free Pascal. В ранних версиях они работали только со строками. Начиная с Free Pascal 3.2 стали поддерживаться одномерные динамические массивы.

Процедура `Insert` вставляет содержимое одной строки или массива в указанную позицию другой строки или массива.

Формат вызова процедуры `Insert`: *Insert(источник, приемник, позиция);*

Процедура `Delete` удаляет определенное количество элементов начиная с указанной позиции. Процедура `Delete` не выполняет никаких действий если индекс или количество являются отрицательными.

Формат вызова процедуры `Delete`: *Delete(цель, позиция, количество);*

В заключении стоит сказать о задании позиции. Элементы строк нумеруются начиная с единицы, а элементы динамических массивов нумеруются с нуля.

Глава 23. Записи

Запись как пользовательский тип данных

Запись объединяет внутри себя переменные разных типов. 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.

Работа с полем внутри экземпляра записи очень проста. 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 с набором особых полей. Особые поля идут сразу после обычных. Доступность особых полей определяется текущим значением обычного поля.

Запись с вариантами определяется следующим образом:

```
Type запись=Record  
поля  
case поле:тип of:  
значение1:поле1;  
...  
значениеN:полеN;  
End;
```

Объявление особого поля имеет следующий вид: *(поле:тип)*

Можно объявить несколько особых полей одного типа перечислив их через запятую. Обращение ко всем полям происходит обычным образом.

Глава 24. Значения по умолчанию

Задание значений по умолчанию

В третьей версии Free Pascal появилась возможность автоматически присвоить переменной значение по умолчанию, основываясь на её типе.

Делается это при помощи следующей конструкции: *переменная:=default(mun);*

Данная конструкция работает с базовыми типами и записями.

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.

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

Глава 25. Упакованные сущности

Выравнивание

Центральный процессор обрабатывает данные, используя машинные слова.

Выравнивание позволяет оптимизировать размещение данных в памяти.

Упакованные сущности хранят данные с учетом выравнивания.

Границы выравнивания задаются соответствующими директивами компилятора.

О директивах компилятора будет рассказано в седьмой части книги.

Упаковка записей

Объявление упакованной записи отличается от объявления обычной записи наличием ключевых слов *packed* или *bitpacked* перед ключевым словом *record*.

При использовании ключевого слова *bitpacked* выравнивание выполняется с учетом отдельных бит.

Упаковка массивов

Объявление упакованного массива отличается от объявления обычного массива наличием ключевых слов *packed* или *bitpacked* перед ключевым словом *array*.

Динамический массив не может быть упакованным.

Процедура *pack* упаковывает содержимое обычного массива и помещает его в упакованный массив, а процедура *unpack* выполняет обратные действия.

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

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

Глава 26. 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 27. 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 переменные, которые имеют допустимый тип. К допустимым типам относятся типы которые были рассмотрены в главе 3 и главе 20.

Глава 28. Файлы

Файловые типы

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

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.

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.

Переопределение стандартного ввода-вывода

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

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

Режим открытия двоичных файлов

По умолчанию подпрограмма Reset открывает файлы для чтения. Режим открытия файлов можно изменить, если присвоить predefined переменной FileMode нужный флаг.

Режим должен быть задан до открытия файла.

Флаг	Режим
0	Чтение
1	Запись
2	Чтение и запись

Двоичные файлы

Типизированные и не типизированные файлы являются разновидностью двоичных файлов.

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

Текстовые файлы

Текстовые файлы относятся к самым простым файлам. 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
Flush(файл)	Форсирует запись содержимого внутреннего буфера в текстовый файл

Глава 29. 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 30. 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 31. Ограничение доступа к глобальным переменным

Free Pascal имеет особенность, которая отсутствует в Delphi. Доступ к глобальным переменным можно ограничить при помощи свойств.

Взаимодействие и передача данных между переменной и свойством происходит при помощи неявного обращения к переменной или вызова подпрограммы.

Ограничения задаются при помощи спецификаторов доступа. Спецификатор *read* дает права чтения. Спецификатор *write* дает права записи.

Объявление свойства осуществляется следующим образом:

```
property свойство: тип read идентификатор1 write идентификатор2;
```

Идентификатор представляет собой имя переменной или подпрограммы, которые привязываются к свойству.

Вы можете использовать сокращенную форму объявления свойства и сделать свойство доступным только для чтения или только для записи.

Глава 32. Потокосые переменные

Потокосые переменные похожи на обычные глобальные переменные, но они видны только в своем потоке.

При объявлении потокосых переменных используется ключевое слово *threadvar* вместо ключевого слова *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. Утверждения

Утверждения нужны для проверки допущений в коде. Работа с утверждениями осуществляется через процедуру *Assert*. Она имеет две формы.

Процедура в первой форме берет логическое выражение в качестве аргумента и аварийно завершает работу если выражение ложно.

Процедура во второй форме берет логическое выражение и строку с сообщением в качестве аргументов. Если выражение ложно, то выводится сообщение и работа программы аварийно завершается.

Процедура *Assert* выполняет проверки только если включена соответствующая директива компилятора.

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.

Используйте функцию ParamCount, чтобы узнать количество переданных параметров. Эта функция не берет аргументов.

Используйте функцию ParamStr, чтобы узнать значение конкретного параметра. Она берет в качестве аргумента номер параметра. Нулевой параметр содержит имя исполняемого файла программы.

Глава 4. Завершение программы

Нормальное завершение

Когда программа завершает свою работу она передает операционной системе код возврата, по которому можно узнать результат работы программы.

Присвойте целочисленное значение глобальной predefined переменной ExitCode, чтобы определить код возврата. Допустимы отрицательные числа.

Адрес ошибки

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

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.

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

Глава 5. Обработка ошибок

Исключения

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

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 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 памятью

Исключение	Описание
EAccessViolation	Ошибка доступа к области памяти
EHearMemoryError	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

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

Исключения, связанные с железом

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

Глава 6. 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.

Имена модулей

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

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.

Альтернативный формат имен модулей

Начиная с третьей версии Free Pascal так же поддерживается 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 описание 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

Имена переменных или констант из интерфейсного раздела становятся недоступными если они совпадают с другими именами. В этом случае используется следующая конструкция: *модуль.имя*

Глава 7. Динамические библиотеки

Пару слов о библиотеках

Динамической библиотекой называют двоичный файл который содержит необходимые процедуры и функции. Динамическая библиотека загружается при запуске программы или во время ее выполнения.

Библиотеки в разных системах

Разные операционные системы имеют разный формат динамических библиотек. Так же в разных системах отличаются расширения имен библиотечных файлов.

Система	Расширение имени файла
Microsoft Windows	.dll
Unix подобные	.so
OS X	.dylib

Загрузка библиотек

Существует два способа загрузки библиотек: статическая загрузка и динамическая загрузка.

При статической загрузке библиотека загружается автоматически при запуске программы.

При динамической загрузке библиотека загружается во время работы программы по требованию программиста.

Вызов подпрограмм при статической загрузке библиотек

Это самый простой вариант использования библиотечных подпрограмм. Для использования библиотечных подпрограмм необходимо объявить прототипы необходимых процедур и функций.

В этих прототипах необходимо использовать вторую форму модификатора *external*. Не забывайте так же указывать в прототипах корректные модификаторы. Иначе библиотека будет работать некорректно.

Вызов подпрограмм при динамической загрузке библиотек

Для динамической загрузки библиотек необходимо использовать модуль `dynlibs`.

Сама задача разбивается на несколько этапов: загрузка библиотеки, получение адресов необходимых подпрограмм, выгрузка библиотеки.

Загрузка библиотеки происходит при помощи функции `LoadLibrary`. Эта функция берет строку с именем файла в качестве аргумента и возвращает идентификатор библиотеки. Этот идентификатор имеет тип `TLibHandle`. При возникновении ошибки возвращается специальное значение `NilHandle`.

Узнать причину ошибки можно при помощи функции `GetLoadErrorStr`, которая не берет аргументов и возвращает строку с описанием ошибки.

Для использования библиотечных подпрограмм необходимо присвоить переменным процедурного типа адреса соответствующих подпрограмм. При определении соответствующих процедурных типов необходимо указывать корректные модификаторы.

Для получения этих адресов используются функции `GetProcAddress` или `GetProcAddress`. Эти функции берут идентификатор библиотеки и строку с именем подпрограммы в качестве аргументов. В случае ошибки эти функции возвращают *Nil*.

Для выгрузки библиотек используются функции `UnloadLibrary` или `FreeLibrary`. Обе функции берут идентификатор библиотеки в качестве аргумента.

Написание библиотек

Структура библиотеки похожа на структуру модуля. Она выглядит следующим образом:

```
library имя;  
uses модули;  
объявления  
подпрограммы  
exports  
список  
end.
```

Рассмотрим эту структуру подробнее.

Вначале указывается имя библиотеки. Затем идет необязательный раздел с подключением необходимых модулей.

Далее идут объявления необходимых переменных и констант.

После этого идут определения необходимых подпрограмм. Для правильной работы библиотеки стоит не забывать указывать необходимые модификаторы подпрограмм.

Заканчивается все списком экспорта. В этом списке перечисляются все подпрограммы которые будут доступны из библиотеки для внешних программ. Каждый элемент списка занимает одну строку. Он представляет собой имя подпрограммы с точкой запятой на конце.

Экспортируемое имя подпрограммы может отличаться от фактического. Это достигается назначением подпрограмме псевдонима. Псевдоним назначается при помощи следующей конструкции: *элемент name 'имя'*;

Библиотека может иметь необязательную секцию инициализации. Код из этой секции выполняется автоматически при загрузке библиотеки.

Секция инициализации располагается в самом конце и выглядит следующим образом:

```
begin  
код  
end.
```


Часть 3. Объектно-ориентированное программирование

Глава 1. Принципы объектно-ориентированного программирования

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

Суть процедурного программирования состоит в разбивке исходной задачи на совокупность связанных подпрограмм. Этот прием называют функциональной декомпозицией.

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 или полями.

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.

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

Один класс может служить основой для другого. Этот механизм называют наследованием.

Наследование является простым при наличии одного родительского класса. Наследование является множественным при наличии нескольких родительских классов

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:

```
Типе 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. Синтаксис объявления члена класса аналогичен синтаксису объявления поля записи.

Потом идут прототипы методов. Прототип метода похож на определение подпрограммы, но у него отсутствует тело.

Запрет наследования

Начиная с Free Pascal 2.4.2 появилась возможность запретить классу быть родителем производных классов при наследовании. О наследовании вы узнаете далее из этой книги.

Запрет наследования осуществляется при помощи ключевого слова *sealed*, которое ставится после ключевого слова *class*. Ключевые слова отделяются друг от друга пробелом.

Статические поля

Данные-члены объявленные с модификатором *static* являются статическими данными. Они похожи на глобальные переменные доступные внутри класса. Они принадлежат всему классу в целом и к ним можно обращаться даже без создания объекта.

Формат задания спецификаторов членов аналогичен формату задания спецификаторов методов.

В Free Pascal 2.6.4 появился альтернативный синтаксис объявления статических полей. Он имеет следующий вид: *class var имя:тип;*

К статическим полям можно обращаться через объекты или напрямую через класс. Прямое обращение имеет следующий синтаксис: *класс.поле*

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. Это ключевое слово является псевдонимом текущего экземпляра.

Приведение типов

Класс является разновидностью определяемых пользователем типов данных. Для приведения объекта к классовому типу используется оператор *as*.

Формат вызова оператора: *объект as класс*;

Конструкторы и деструкторы

Каждый класс должен содержать специальные методы: конструктор и деструктор. Конструктор вызывается при создании объекта. Деструктор вызывается при уничтожении объекта.

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

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

Использование конструктора и деструктора обязательно.

Конструкторы и деструкторы имеют несколько отличий от обычных методов. Они заключаются в следующем:

- Каждый объект может иметь только один конструктор и деструктор
- Имена конструкторов и деструкторов могут быть произвольны
- Конструкторы и деструкторы не имеют параметров
- Конструкторы и деструкторы не могут быть виртуальными

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. Объект размещается в динамической памяти.

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.

Если объект объявлен, но не создан, то оператор возвращает *False*. На практике оператор *is* часто используется вместе с условным оператором *if*.

Разновидности методов

Существуют несколько разновидностей методов. Вид метода определяется при помощи добавления спецификатора в прототип метода.

Синтаксис: *прототип(параметры); спецификатор;*

Спецификаторы должны идти друг за другом если необходимо использовать несколько спецификаторов одновременно.

Список спецификаторов дан в таблице ниже.

Спецификатор	Описание
Virtual	Виртуальный метод
Abstract	Абстрактный метод
Dynamic	Синоним Virtual

Метод является обычным, если явно не указан его спецификатор.

Методы обработки сообщений

При взаимодействии с графическим интерфейсом программе отправляются специальные уведомления, которые называют сообщениями. Для их обработки нужны особые методы.

Прототипы таких методов маркируются спецификатор *message*.

Формат использования спецификатора: *message идентификатор*

Идентификатором является константа, которая определяет тип сообщения.

Статические методы

Статические методы появились во Free Pascal 2.2. Это разновидность методов принадлежит всему классу в целом. Их можно вызывать даже не создавая объект. Такие методы имеют несколько ограничений.

Ограничения статических методов:

1. Нельзя использовать ключевое слово *Self*.
2. Метод не может быть абстрактным, виртуальным или перегружаемым.
3. Нельзя обращаться к свойствам или обычным данным-членам.
4. Нельзя обращаться к обычным методам.

В прототипах и определениях таких методов перед ключевым словом *procedure* или *function* ставиться ключевое слово *class*.

В третьей версии Free Pascal появилась возможность добавлять к прототипам статических методов модификатор *static*. Это сделано для повышения совместимости с современными версиями Delphi.

Статические методы могут вызываться через объекты или напрямую через класс.

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 так же могут быть использованы для ограничения доступа к членам класса. Объявление свойства в классе аналогично объявлению обычного свойства. Привязываемые к свойству член или метод должны быть ранее объявлены в классе.

Статические свойства

Статические свойства работают исключительно со статическими полями и методами. Их объявление отличается от объявления обычных свойств наличием ключевого слова *class* перед ключевым словом *property*.

Статические свойства появились во Free Pascal 2.6.4.

Переопределение свойств

Свойства базового класса могут быть переопределены в производном классе. При переопределении свойство привязывается к члену или методу производного класса.

Свойства по умолчанию

Свойства по умолчанию позволяют использовать сокращенную форму для присваивания свойству значения. В каждом классе может быть только одно свойство по умолчанию.

Формат сокращенной формы: *объект:=значение;*

Свойства по умолчанию маркируются модификатором *Default*.

Свойства-массивы

Свойства-массивы передают в привязанные методы целочисленный индекс.

Индекс может быть одномерным или двумерным. Поэтому синтаксис объявления свойства имеет две формы.

В случае с одномерным индексом синтаксис имеет следующий вид:

property свойство[индекс:mun]:mun read метод1 write метод2;

Для двумерного индекса объявление свойства принимает следующий вид:

```
property свойство[индекс1,индекс2:тип]:тип read метод1 write метод2;
```

В первом случае методы читающие данные должны иметь один целочисленный параметр, а методы записывающие данные должны иметь два параметра. Первый параметр метода записи должен быть целочисленный.

Во втором случае методы читающие данные должны иметь два целочисленных параметра, а методы записывающие данные должны иметь три параметра. Первые два параметра метода записи должны быть целочисленными.

Свойства-массивы могут быть свойствами по умолчанию.

Индексные свойства

Взаимодействие с членами в индексных свойствах происходит через неявный вызов методов. Индексные свойства содержат внутри себя константное значение, которое автоматически передается в метод при обращении к свойству.

Синтаксис объявления свойства выглядит следующим образом:

```
property свойство:тип index значение read метод1 write метод2;
```

Первым параметром у метода должен быть целочисленный параметр.

Глава 4. Наследование

Взаимоотношения с родителями

Родительским называется класс от которого может порождаться другой класс., Наследование является простым если класс имеет одного родителя.

При наследовании конструкция описывающая класс меняется. Сразу после ключевого слова *class* идет следующая конструкция: (родитель)

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 методом.

Если вы хотите чтобы перегруженный метод являлся обычным методом, то замените модификатор *override* на *reintroduce*.

Абстрактные методы

Абстрактные методы не имеют реализации в базовом классе. Свою реализацию такие методы получают в производном классе.

Вызов метода родительского класса

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

Первая форма: *inherited*;

Вторая форма: *inherited метод(аргументы)*;

Первая форма используется когда имена методов в родительском и базовом классе совпадают. В этом случае методу передаются те же аргументы, что и методу дочернего класса.

Во второй форме метод и передаваемые ему аргументы задаются явно.

Глава 5. Интерфейсы

Поддержка интерфейсов

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

Можно так же определять объекты без использования классов. Существует несколько отличий от представителей классов.

- Нельзя использовать свойства.
- По умолчанию объект размещается в стеке.
- Память под объект выделяется при входе в блок, который его содержит.
- Память автоматически освобождается при выходе из блока.
- Конструктор и деструктор необходим только для динамических объектов или при наличии виртуальных методов.

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. Объявление экземпляра объекта отличается от объявления обычной переменной указанием объекта вместо стандартного типа.

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.

Удобная работа с объектами

Оператор *with* позволяет более удобно манипулировать экземпляром объекта.

Синтаксис:

```
with экземпляр do  
begin  
код  
end;
```

В коде который заключен между ключевыми словами *begin* и *end* не требуется использование имени объекта для работы с его членами и методами.

В качестве экземпляра может выступать представитель класса или обычный объект. Так же этот оператор 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*.

Упакованные объекты

Данные-члены упакованных объектов выравниваются по границе в один байт.

Синтаксис описания упакованных объектов аналогичен синтаксису описания обычных объектов, но ключевому слову *object* предшествует ключевое слово *packed*.

Глава 2. Динамические объекты

Особенности динамических объектов

Использование обычных объектов часто влечет за собой большое потребление памяти. Поэтому так же предусмотрена возможность 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 впервые появились в Delphi 2005.

Расширенные записи доступны если включен режим совместимости с Object Pascal и задействована соответствующая директива компилятора.

Расширенные записи поддерживаются начиная с Free Pascal версии 2.6.

Расширенные записи похожи на классы, но имеют ряд отличий, которые перечислены ниже:

1. Не поддерживается наследование
2. Нельзя использовать свойства
3. Запрещено использовать ключевые слова *published* и *protected*
4. Статические методы должны быть маркированы модификатором *static*
5. Конструкторы и деструкторы не используются
6. Запрещены абстрактные и виртуальные методы
7. 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 классам.

Часть 5. Обобщения

Глава 1. Шаблоны

Универсальность шаблонов

Шаблоны позволяют создавать универсальный код, который может работать с разными типами данных. Построение варианта, который работает с конкретным типом, производится компилятором.

Шаблоны появились во Free Pascal версии 2.2.

Описание шаблонного класса

Описание шаблона похоже на описание обычного класса. Синтаксис описания шаблона имеет следующий вид:

```
Type generic шаблон<идентификатор> = class  
элементы  
end;
```

Идентификатор представляет собой ложный тип данных. Этот тип можно использовать при объявлении параметров в прототипах и определениях методов шаблона. Ложный тип заменяется на один из реальных типов во время специализации.

Специализация

Построение специализированного класса на основе шаблона называют специализацией.

Синтаксис: *Type класс = specialize шаблон*<тип>;

Построение специализированного класса при объявлении объекта

Начиная с третьей версии Free Pascal появилась возможность выполнять построение специализированного класса при объявлении объекта.

Синтаксис: *Var объект: specialize шаблон*<тип>;

Использование нескольких абстрактных типов

Идентификатор в описании шаблона может быть заменен списком идентификаторов. Идентификаторы в списке разделяются запятой. При специализации так же необходимо заменить тип списком типов.

Работа со специализированными классами

Работа с классами построенными на основе шаблонов аналогична работе с обычными классами.

Глава 2. Обобщенные расширенные записи

Шаблоны так же могут быть и у расширенных записей. Описание такого шаблона аналогично описанию обычного шаблона, но вместо ключевого слова *class* используется *record*.

Специализация записи и работа с экземпляром обобщенной записи так же производятся аналогично.

Обобщенные расширенные записи появились в третьей версии Free Pascal.

Часть 6. Помощники

Глава 1. Предназначение помощников

Данная реализация Паскаля поддерживает помощники. Помощник позволяет расширить функциональность существующего кода. Основным их достоинством является 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. Использование помощников требует задействования соответствующих директив компилятора.

Глава 2. Помощники классов

Ограничения

Помощники классов появились во Free Pascal версии 2.6. Они не являются полноценными классами и имеют 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 обычных 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 появились во Free Pascal версии 2.6. Они 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. 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 соответствующую директиву

В остальном они ведут себя как обычные помощники.

Помощники простых типов появились в третьей версии Free Pascal.

Часть 7. 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: $\{\$директива\} \text{ параметры}$

Глава 2. 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 3. 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 предполагается расширение .pp.

Связанные между собой файлы рассматриваются при компиляции как одно целое.

Глава 4. 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 5. Типы программ

Тип программы создаваемой программы определяется флагом глобальной директивы *APPTYPE*. Допустимые флаги приведены ниже.

Флаг	Описание
CONSOLE	Программа с текстовым интерфейсом
GUI	Программа с графическим интерфейсом
FS	Полно-экранная программа для OS/2
TOOL	Служебная утилита для Mac OS

Глава 6. 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 необязательны.

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 необязательны.

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 7. 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 в начале файла с исходным кодом.

Директива	Параметры	Описание
APPTYPE	Тип	Определяет тип создаваемой программы
ASSERTIONS	ON или OFF	Включает или выключает проверку утверждений
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	Задает минимальный размер стека в байтах
MAXSTACKSIZE	Размер	Задает максимальный размер стека в байтах
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	Режим	Устанавливает режим совместимости
MODESWITCH	Директива	Выборочно задействует те или иные возможности данного диалекта
PROFILE	ON или OFF	Определяет наличие информации для профайлера
EXTENDED SYNTAX	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	Оптимизация	Задаёт режим оптимизации
PACKETSET	Размер	Определяет количество байт для хранения множеств
ALIGN	Граница	Задаёт границу выравнивания данных в байтах
PACKETRECORDS	Граница	Задаёт границу выравнивания в байтах для записей
CALLING	Формат	Определяет формат передачи параметров по умолчанию
BITPACKING	ON или OFF	Включает или выключает упаковывание с учётом отдельных бит
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
WRITEABLECONST	ON или OFF	Разрешает или запрещает модификацию типизированных констант
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 с файлами
I-	Нет	Выключает выброс исключений при ошибках с файлами
WAIT	Нет	Приостанавливает компиляцию и ждёт нажатия клавиши
HINT	Текст	Выводит на экран подсказку
WARNING	Текст	Выводит на экран предупреждение

Глава 8. Избирательное задействование языковых возможностей

Пришло время подробно рассмотреть флаги директивы *MODESWITCH*. Список основных флагов дан в таблице ниже.

Флаг	Описание
CLASS	Разрешает использование классов
OBJPAS	Автоматически подключает модуль ObjPas
RESULT	Разрешает использование ключевого слова Result
PCHARTOSTRING	Автоматическое преобразование строк в Си стиле в обычные
CVAR	Разрешает использование ключевого слова Cvar
NESTEDCOMMENTS	Разрешает использование комментариев из нескольких строк
CLASSICPROCVARS	Включает классические процедурные переменные
MACPROCVARS	Разрешает использование переменных в стиле Macintosh
REPEATFORWARD	Реализация и предварительное объявление должны совпадать
POINTERTOPROCVAR	Автоматическое преобразование указателей в процедурный тип
AUTODEREF	Автоматическое преобразование типизированных указателей
INITFINAL	Разрешает секции инициализации и завершения
ANSISTRINGS	Разрешает использование обычных строк
OUT	Разрешает использование ключевого слова Out
DEFAULTPARAMETERS	Разрешает использование параметров по умолчанию
HINTDIRECTIVE	Разрешает использование указательных директив
DUPLICATELOCALS	Разрешает совпадение локальных переменных метода и свойств класса
PROPERTIES	Разрешает использование свойств
ALLOWINLINE	Разрешает использование выстраиваемых подпрограмм
EXCEPTIONS	Разрешает использование исключений
ADVANCEDRECORDS	Разрешает использование расширенных записей
UNICODESTRINGS	Включает в строках использование Юникода по умолчанию
TYPEHELPERS	Разрешает использование помощников

Глава 9. Режимы совместимости

Переключение между режимами

Переключение между режимами осуществляется с помощью директивы *MODE*.

Флаг	Режим
Default	Режим по умолчанию. Аналогичен режиму Free Pascal
FPC	Режим Free Pascal
TP	Режим совместимости с Turbo Pascal
Delphi	Режим совместимости с Delphi
DELPHIUnicode	Режим совместимости с современными версиями Delphi
EXTENDED Pascal	Режим совместимости с Extended Pascal
OBJFPC	Режим совместимости с Object Pascal
MAC	Режим совместимости с Mac Pascal
ISO	Режим совместимости с ISO 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 прототип должен полностью совпадать с определением.
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 Extended Pascal:

1. Данный режим доступен только во Free Pascal 3.2 и более новых версиях.
2. Данный режим обеспечивает совместимость с диалектом определенном в стандарте ISO 10206.

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 прототип должен полностью совпадать с определением.
3. При перегрузке подпрограмм должен использоваться модификатор *Overload*.
4. Запрещены вложенные комментарии.
5. Модуль ObjPas загружается автоматически.
6. Тип *Integer* эквивалентен типу *LongInt*.
7. Тип *String* по умолчанию эквивалентен типу *AnsiString*.

Особенности режима совместимости с современными версиями Delphi:

1. Обеспечивает максимальную совместимость с современными версиями Delphi.
2. Тип *String* по умолчанию эквивалентен типу *UnicodeString*.

Особенности режима совместимости с Object Pascal:

1. Вы должны использовать оператор взятия адреса для присваивания значений переменным процедурного типа.
2. При использовании предварительного объявления подпрограмм прототип должен полностью совпадать с определением.
3. Разрешена перегрузка подпрограмм.
4. Разрешены вложенные комментарии.
5. Модуль ObjPas загружается автоматически.
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 Turbo Pascal:

1. Перечисление занимает в памяти один байт, если в нем меньше 257 элементов.
2. Вы не можете использовать оператор взятия адреса для присваивания значений переменным процедурного типа.
3. При использовании предварительного объявления подпрограмм прототип должен полностью совпадать с определением.
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 8. 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 интерфейс.

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. Графического интерфейс придумали в корпорации Xerox. Идея возникла в 1970 году и спустя некоторое время 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. Переключатель представляет собой круг с текстом напротив.

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 вертикальными или горизонтальными полосами прокрутки предназначен для хранения элементов, которые не вмещаются в окно. Полосы прокрутки 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

Горизонтальные или вертикальные бегунки позволяют выбрать значение из заданного диапазона.

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

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

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

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 является 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

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

События

Событие	Описание
OnClick	Щелчок на бегунке
OnChange	Изменение позиции бегунка

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

Поле для ввода и редактирования текста из нескольких строк

Поле для ввода и редактирования текста из нескольких строк является объектом класса TМемо. 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 TString	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 всплывающее вызывается по щелчку правой кнопкой мыши. Главное меню является объектом класса TMainMenu. Всплывающее меню является объектом класса TPopupMenu. Они оба используют массив объектов класса TMenuItem 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 TMenuItem.

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

Чтобы привязать всплывающее 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. Воспользуйтесь строковым свойством Caption объекта класса TTabSheet, чтобы задать заголовок вкладки.

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 TScrollBar. Логическое свойство AutoScroll скрывает или показывает полосы прокрутки.

Глава 4. Диалоги

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.

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	Заккрытие диалога без выбора файла
OnCanClose	Заккрытие диалога с выбором файла

Диалог выбора каталога

Диалог выбора каталога является объектом класса TSelectDirectoryDialog. Этот диалог вызывается 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	Заккрытие диалога без выбора каталога
OnCanClose	Заккрытие 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

Программисту бывает иногда нужно выполнять код через определенные интервалы времени. Для этой цели существуют таймеры. 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 TProcess.

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	Задаёт 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
ProcessID	Целое число	Идентификатор процесса

Часть 9. Среда Lazarus

Глава 1. Средства быстрой разработки

В настоящее 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.

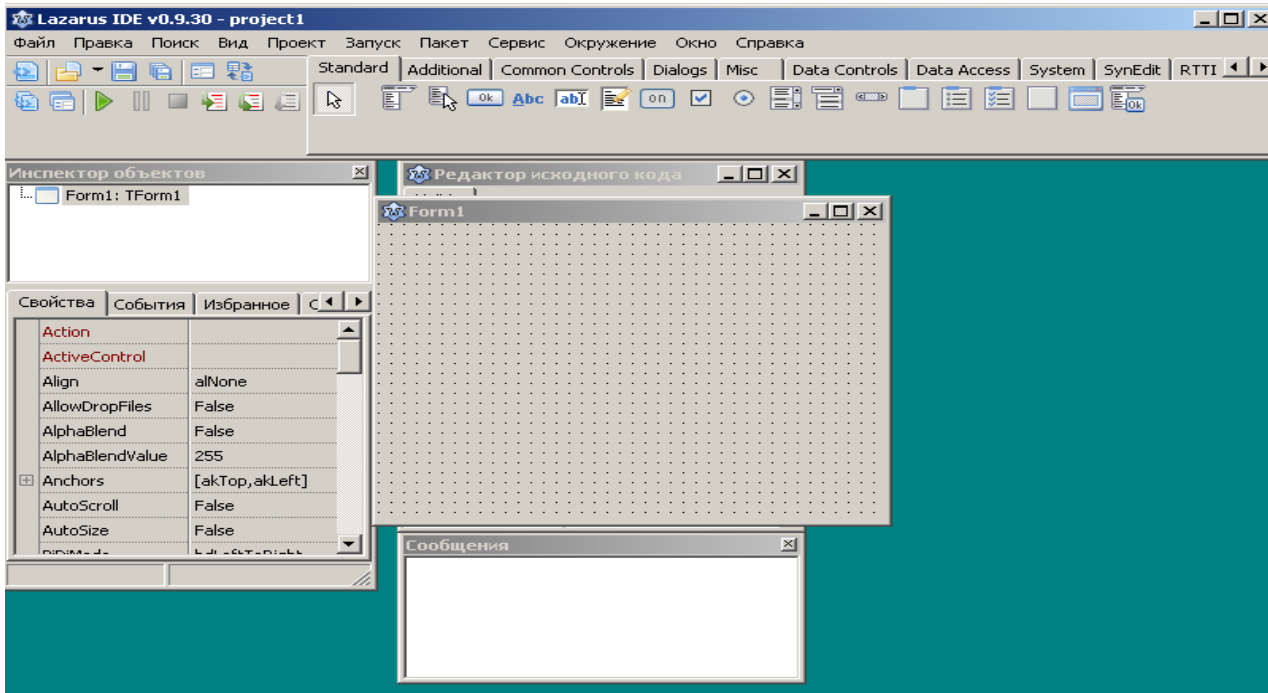
Расширение файла	Описание
pas	Исходный код на паскале
lpr	Исходный код главного модуля
lfm	Описание формы
lpi	Описание проекта

Содержимое главного модуля и файлов описаний генерируется автоматически.

Глава 3. Проектирование в Lazarus

При запуске Lazarus автоматически создает новый проект.

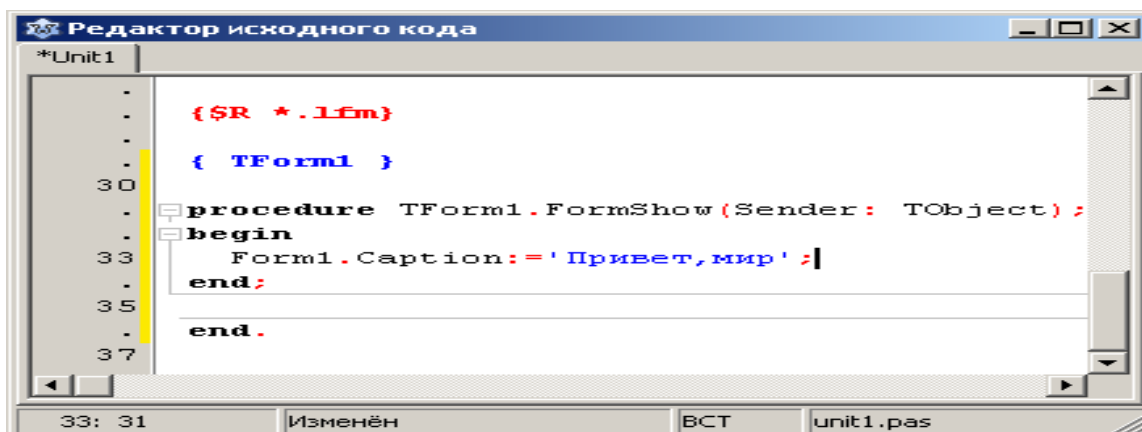
Вы увидите следующие окно:



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. Поэтому вам необходимо будет самостоятельно углублять сведения, полученные из этой книги.

Вокруг рассмотренного в этой книге языка и среды сложилось обширное сообщество. 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. Русскоязычное сообщество имеет свой сайт в интернете. На нем вы найдете полезные статьи по Free Pascal и Lazarus, а так же активный форум. Адрес сайта — <http://freepascal.ru>

В этой книге я постарался изложить основные сведения необходимые вам для успешного старта. 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 программированию / Е.Р.Алексеев, О.В.Чеснокова, Т.В.Кучер — М. : АЛТ 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 C++: 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