Оглавление

Введение ... 3
Раздел 1. Турбо Паскаль ... 6
  Глава 1. Знакомство с языком и интегрированной средой ... 6
    1.1 Среда Турбо-Паскаль ... 6
    1.2 Отладка программ пользователя в Turbo Pascal ... 11
  Глава 2. Основные элементы языка ... 19
    2.1 Алфавит языка и структура программы ... 19
    2.2 Арифметические операции и стандартные функции ... 21
    Глава 3. Простые типы данных ... 24
  Глава 4. Операторы языка: простые, структурированные ... 30
    4.1 Простые операторы ... 30
    4.2 Структурные операторы ... 31
  Глава 5. Составные, или структурированные, типы данных ... 35
    5.1 Массивы ... 35
    5.2 Символьные и строковые величины ... 37
    5.3 Записи и множества ... 40
    5.4 Файловые типы ... 44
    5.5 Совместимость и преобразование типов ... 49
    5.6 Динамическая память ... 52
  Глава 6. Процедуры и функции ... 53
    6.1 Ввод и вывод данных ... 53
    6.2 Процедуры ... 55
    6.3 Функции ... 56
    6.4 Рекурсия ... 56
  Глава 7. Стандартные библиотечные модули ... 57
    7.1 Модуль Crt ... 58
    7.2 Графика в Турбо Паскале ... 59
    7.3 Вывод простейших фигур ... 63
    Стандартные стили заполнения ... 66
  Глава 8. Объектно-ориентированное программирование ... 70
    8.1 Создание объектов ... 72
    8.2 Использование объектов ... 81
Раздел 2. Delphi ... 86
  Глава 1. Знакомство со средой Delphi ... 86
    1.1 Общие представления ... 86
    1.2 Основные отличия различных версий Delphi ... 87
  Глава 2. Основы визуального программирования ... 92
    2.1 Знакомство с компонентами ... 92
    2.2 Свойства в Delphi ... 114
    2.3 Методы в Delphi ... 115

Глава 3. Обработка исключительных ситуаций в Delphi 118
3.1 Структурная обработка исключительных ситуаций 118
3.2 Предопределенные обработчики исключительных ситуаций....... 121
Глава 4. События в Delphi 125
Глава 5. Средства создания мультимедийных приложений 129
Глава 6. Введение в Object Pascal.. 130
6.1 Структура программ Delphi... 130
6.2 Элементы программы 135
6.3 Операторы языка 139
Глава 7. Формы 146
7.1 Разновидности форм 146
7.2 Создание и использование форм 153
Глава 8. Среда разработчика.. 155
8.1 Главное меню.. 155
8.2 Работа с редактором.. 172
8.3 Отладка программ 176
Библиография 180

Введение

> «Паскаль — очень элегантный язык. Он по-прежнему жив. Он породил немало своих последователей и оказал глубокое воздействие на проектирование языков».
> Денис Ритчи.:

К началу 60-х годов количество основных языков программирования стало резко возрастать. Поэтому были предприняты попытки создать универсальный язык программирования, но ни одна из этих попыток не увенчалась успехом. Среди десятка наиболее распространенных на тот период времени языков программирования каждый был ориентирован на решение определенных задач. Бейсик употреблялся для написания простых программ. Фортран - с его четко определенными правилами выполнения арифметических операций - являлся классическим языком программирования для решения математических и физических задач. Язык программирования COBOL был задуман как основной язык для массовой обработки данных в сферах управления и бизнеса. Другие языки программирования были также специализированы. Еще один язык программирования - Алгол - предназначался для записи алгоритмов, которые строятся в виде последовательности процедур, применяемых для решения поставленной задачи. Программисты далеко неоднозначно приняли Алгол, широкого одобрения он не получил. Но все же влияние Алгола на развитие других языков программирования оказалось значительным.

Среди языков, целью создания которых было улучшение Алгола, следует особо отметить Паскаль, разработанный в конце 60-х годов швейцарским ученым Никлаусом Виртом. Pascal был назван в честь французского философа и математика XVII века Блеза Паскаля. Некоторое время Никлаус Вирт был профессором информатики в Федеральном техническом университете в Швейцарии и нуждался в языке, с помощью которого относительно легко можно было бы обучать студентов навыкам программирования на хорошем уровне. Базовая концепция Паскаля была разработана Виртом примерно в 1970 году, и Паскаль очень быстро начал повсеместно распространяться, прежде всего, благодаря легкости в изучении и наглядности написанных на нем программ.

Язык Паскаль требовал от программиста определения всех переменных в отдельной секции в начале программы. Так как эти определения задавались явным образом, то в программах появлялось сравнительно немного ошибок, и их было проще понять и исправить разработчику. Это сделало Паскаль популярным при создании больших программ. Вскоре он был

объявлен официальным языком программирования для учащихся средних школ, которые намерены специализироваться в области вычислительной техники и программирования в американских университетах.
В то же время, стандартный Паскаль обладал рядом существенных недостатков. Поэтому Вирт продолжил развивать свое детище, так через девять лет, в 1979 году, появилась Модула-2, прежде всего от Паскаля она отличалась тем, что давала возможность использовать модульное программирование, а значит, с ее помощью уже можно было создавать достаточно большие проекты. К середине 70-х годов назрела необходимость разработать международный стандарт на Паскаль. В результате, в 1982 году появился стандарт ИСО 7185. Но Вирт не останавливался на достигнутом, и немного позднее появились Ада и Оберон, которые позволяли использовать типы и объекты, что уже давало кардинально новые возможности для разработчиков.
Здесь нужно заметить, что стандартный Паскаль был действительно очень простым языком. А Turbo Pascal включает в себя многие положительные качества, как Модулы, так и Oberon"а, и является гораздо более сложным и мощным языком, чем Паскаль Вирта. Он позволяет использовать и модули, и типы, и даже объекты, давая всю мощь объектно-ориентированного программирования в руки разработчика. Андерс Хейлсберг является отцом-создателем Турбо Паскаля, вскоре вытеснившего с рынка все другие спецификации. Это победоносное шествие началось в 1984 году с появлением версии Turbo Pascal 2.0, ее распространение шло стремительными темпами. Версии 3.0, 4.0 и 5.0 выходили соответственно в 1985 и 1988 годах (последние две). В них появилось несколько революционных нововведений: возможность разбивать программу на несколько файлов (модулей), интерфейс взаимодействия с MS-DOS и встроенный отладчик. Отладчик (debugger) - это очень полезное средство, позволяющее в период выполнения программы просматривать содержимое регистров процессора, текущие значения переменных и последовательность выполнения программы. От версии к версии оптимизировалась работа компилятора, который генерировал исполняемый код на основе текста программы. Даже достаточно простые программы, перенесенные с других языков программирования, не использующие всех возможностей, которые давал Паскаль, после компиляции начинали работать существенно быстрее. Компиляция происходила в полтора раза быстрее, файл программного кода получался во много раз меньше, чем у Бейсика, а скорость выполнения полученного кода просто впечатляла. Сейчас, когда машины оснащаются сотнями мегабайт оперативной памяти и быстродействующими процессорами, мало кто обращает внимание на такие, казалось бы, мелочи, а тогда, при работе на 8088 процессоре со всего 640 кб оперативной памяти, это было очень важно.

Последующее развитие Pascal привело к появлению таких библиотек, как Turbo Vision, с использованием которой написан известный Dos Navigator, а также языка Object Pascal, который впоследствии стал основой для создания Delphi.
Delphi - компилятор языка Object Pascal. Delphi 1 был первым инструментарием разработки Windows-приложений, объединившим в себе оптимизирующий компилятор, визуальную среду программирования и мощные возможности работы с базами данных. Годом позже Delphi 2 предложил все то же, но на новом уровне современной 32-битной операционной системы Windows 95 и Windows NT. Кроме того, Delphi 2 предоставил программисту 32-битовый компилятор, создававший более быстрые и эффективные приложения, мощные библиотеки объектов.
Продолжительная работа команды разработчиков Delphi привела к появлению в третьей версии продукта расширенного набора инструментов для создания приложений, возможности использования технологий COM для разработки приложений WWW и многих других современных технологий программирования.
Delphi 8 является очередным шагом в эволюции компиляторов Паскаля. Пожалуй, мало найдется языков, которые бы жили так долго, при этом оставаясь актуальными. Действительно, Delphi позволяет даже самому неискушенному в программировании человеку после сравнительно небольшого времени, потраченного на его изучение, создавать профессионально выглядящие программные продукты с графическим интерфейсом пользователя в стиле Windows. Особенно удобно на Delphi работать с базами данных. Естественно, эта легкость работы с Delphi оборачивается некоторыми минусами языка. Основной недостаток Delphi - скорость работы, она очень и очень невысока. Да и генерируемый компилятором код получается слишком большим. Так, создание пустого окна даст exe-файл размером почти в полмегабайта, на C++ - такое же займет 15-30 килобайт. Тем не менее, для некоторых сфер применения Delphi является идеальным решением, поэтому сейчас он достаточно популярен.
Таким образом, в настоящее время действуют три стандарта языка. Первый из них - нерасширенный Паскаль (unextended Pascal) был разработан в 1983 году и практически полностью совпадает с описанием языка в нотации Вирта. Второй - Extended Pascal - содержит расширения, касающиеся модульного программирования (раздельная компиляция модулей, импорт-экспорт подпрограмм, интерфейсная часть и реализация), и дополнен рядом процедур и функций (прямой доступ к файлам, работа со строками и т д.) Последний - объектный Паскаль (Object-Oriented Extensions to Pascal), в отличие от первых двух, формально не утвержден, но оформлен в виде отчета в 1993 г.Объектный Паскаль поддерживает классы, обладающие свойствами и методами, наследование классов, переопределение методов у по-

томков (полиморфизм) и ряд других атрибутов объектно-ориентированного программирования.

Раздел 1. Турбо Паскаль

Глава 1. Знакомство с языком и интегрированной средой

1.1 Среда Турбо-Паскаль

В 1992 году фирма Borland International выпустила два пакета программирования, основанные на использовании языка Паскаль – Borland Pascal 7.0 и Turbo Pascal 7.0.

Пакет Borland Pascal 7.0 включает в себя три режима работы: в обычном режиме операционной системы MS DOS, в защищенном режиме MS DOS и в среде Windows.

Пакет Turbo Pascal 7.0 позволяет работать только в обычном режиме MS DOS и включает в себя как язык программирования – одно из расширений языка Паскаль для IBM PC, так и среду, предназначенную для написания, отладки и запуска программ. Этот язык характеризуется расширенными возможностями по сравнению со стандартом, библиотекой модулей, позволяющих использовать возможности операционной системы, организовывать ввод-вывод, формировать графические изображения и т.д.

Среда программирования позволяет создавать тексты программ, компилировать их, находить ошибки, компоновать программы из отдельных модулей, выполнять отлаженную программу. Версия 7.0 обеспечивает многооконный режим работы, использование манипулятора «мышь», позволяет применять элементы объектно-ориентированного программирования, обладает встроенным ассемблером, а также инструментальным средством для создания интерактивных программ – Turbo Vision.

Особенности Turbo Pascal 7.0:

- Выделение цветом различных элементов исходного текста программы – идентификаторов, зарезервированных слов, комментариев, строк, чисел, что позволяет на стадии ввода исходного текста устранить многие ошибки;
- Многофайловая система помощи;
- Наличие локального меню, зависящего от текущего состояния среды и вызываемого нажатием либо правой кнопки «мыши», либо комбинацией клавиш Alt+F10;
- Ряд дополнительных расширений языка: использование открытых массивов, параметров-констант, типизированного адресного оператора;
- Наличие дополнительных стандартных процедур и функций;

- Наличие дополнительных ключей компилятора;
- Расширенные возможности объектно-ориентированного программирования;
- Использование кодового сегмента для размещения некоторых видов констант, удаление пустых строк, проверка переполнения величин целых типов, повышающих эффективность кода программ;
- Усовершенствование программы Turbo Vision;
- Улучшенная компоновка системы меню.

Система программирования Turbo Pascal 5.5 (7.0) представляет собой интегрированную среду, включающую в себя:

1. Экранный редактор.
2. Компилятор входного языка.
3. Редактор связей.
4. Интерактивный символьный отладчик.
5. Справочную систему.

Указанные компоненты в совокупности обеспечивают поддержку полного цикла разработки программ на языке Turbo Pascal от этапа задания до формирования готового программного продукта.

Запуск Turbo Pascal производится командой turbo в командной строке.

Для загрузки среды Турбо−Паскаль запускается файл **turbo. exe.**

После запуска Turbo Pascal набирается программа с учетом всех знаков и порядка строк. Затем ее необходимо запустить. Это реализуется нажатием клавиш **Ctrl+F9**. Если программа набрана правильно, то окно редактора чуть-чуть мигнет.

Если есть ошибка, то Паскаль не выполнит программу, а остановится, переместив курсор в строку, где совершена ошибка, и написав, что конкретно за ошибка произошла.

Меню Турбо Паскаля

Верхняя строка содержит «меню» возможных режимов работы Турбо Паскаля, нижняя - краткую справку о назначении основных функциональных клавиш. Вся остальная часть экрана принадлежит окну редактора, очерченному двойной рамкой и предназначенному для ввода и коррекции текстов программ. В его верхней строке приводятся имя того дискового файла, откуда был прочитан текст программы (новому файлу присваивается имя NONAME00.PAS), два специальных поля, используемых при работе с устройством ввода «мышь» (эти поля выделены квадратными скобками), и цифра 1 - номер окна. В Турбо Паскале можно работать одновременно с несколькими программами (или частями одной крупной программы), каждая из которых может располагаться в отдельном окне редактора. Среда позволяет использовать до 9-ти окон редактора одновременно.

Кроме окна (окон) редактора в Турбо Паскале используются также окна отладочного режима, вывода результатов работы программы, справочной

службы, стека, регистров. По желанию они могут вызываться на экран поочередно или присутствовать на нем одновременно.
Меню активизируется нажатием клавиши F10. Перемещение по пунктам меню осуществляется курсорными клавишами. Для выбора пункта меню выделить его курсором и нажать Enter.
Опишем некоторые пункты меню, используемые в Turbo Pascal :

File	**работа с файлами**
New	создать новый файл. Для переименования безымянного файла (NONAME00. PAS) нажать F2 и ввести имя файла.
Open F3	открыть (создать) файл для редактирования. Написать в строке "Name" имя файла и нажать Enter, либо, нажав Tab или Enter, выбрать файл из списка. Для смены каталога выбрать ". . \ ";
Save F2	сохранить файл (записать на диск);
Save as...	сохранить файл под новым именем (исходный файл остается);
Save all	сохранить все файлы в открытых окнах;
Exit ALT+X	выход из среды Турбо Паскаль;
Window	**работа с окнами**
Tile	параллельное размещение окон на экране;
Cascade	последовательное размещение окон в виде каскада;
Close all	закрыть все
Refresh display	обновить (восстановить) экран среды;
Size Ctrl+F5	изменение размеров окна (Shift+курсорные клавиши);
Move Ctrl+F5	перемещение активного окна курсорными клавишами;
Zoom F5	распахнуть окно во весь экран, F5 - для отмены;
Next F6	последовательная смена активного окна;
Previous Shift+F6	смена активного окна в обратном направлении;
Close Alt+F3	закрыть активное окно;
List Alt+0	показать список окон. Для активизации окна выбрать курсором имя окна и нажать Enter.
Edit	**редактирование файла (наибольший размер файла 1Мб)**
Undo Alt+BkSp	отменить предыдущую команду редактирования;
Redo	восстановить отмененную команду редактирования;

Cut Shift+Del	удалить блок с экрана в буфер (в окно Clipboard);
Copy Ctrl+Ins	скопировать блок с экрана в буфер;
Paste Shift+Ins	извлечь (скопировать) блок из буфера на экран;
Clear Ctrl+Del	удалить блок на экране;
Show Clipboard	показать окно для редактирования содержимого буфера.

Основные функциональные клавиши Turbo Pascal :
F1 - получение помощи.
F2 - запись на диск текущего файла.
F3 - загрузка файла с диска.
F4 - выполнение фрагмента программы до строки, содержащей курсор.
F5 - управление размером активного окна.
F6 - переключение между окнами.
F7 - пошаговое исполнение программы, включая вызовы процедур.
F8 - пошаговое исполнение программы, без выходов в процедуры.
F9 - запуск и компиляция программы.
F10 - вызов главного меню.
Esc - выход из меню.
Alt-F9 - компиляция программы.
Ctrl-F9 - запуск (прогон) программы.
Alt-X - выход из среды программирования Turbo Pascal в DOS.
Кроме того, для редактирования программы используются следующие комбинации клавиш:

1. Поместить выделенный текст в буфер (Edit,Copy) - Ctrl-Insert
2. Извлечь текст из буфера (Edit,Paste) - Shift-Insert
3. Вырезать выделенный текст (Edit,Cut) - Shift-Delete.

Для создания файла в TP 7.0 следует указать в Меню File мышью Save As (сохранить как), выбрать каталог, задать имя файла, затем OK. Файл получит расширение .pas.
Чтобы удалить строку, надо нажать Ctrl-Y, подводя курсор к строке.
Таким образом, при компиляции программы (F9), если нет ошибок, текстовый файл программы name.pas преобразуется в двоичный файл с тем же именем, но с расширением exe (name.exe). Этот exe файл уже можно запускать отдельно от среды программирования Turbo Pascal. Если в тексте программы есть ошибки, то компиляция программы будет производиться только после устранения всех ошибок, то есть после отладки программы.
Прервать выполнение программы можно, нажав комбинацию клавиш **Ctrl+Break**. После выполнения программы восстанавливается среда Турбо−Паскаль. Результаты вывода на экран можно посмотреть командой **Alt+F5**.

Справочная служба Турбо Паскаля

Неотъемлемой составной частью среды Турбо Паскаля является встроенная справочная служба. В затруднительной ситуации достаточно нажать F1 и на экране появится необходимая справка. Эта справка зависит от текущего состояния среды, поэтому справочную службу называют контекстно-чувствительной. Например, если нажать F1 в момент, когда среда обнаружила ошибку в программе, в справке будут сообщены дополнительные сведения о причинах ошибки и даны рекомендации по ее устранению.

Существуют четыре способа обращения к. справочной службе непосредственно из окна редактора:

F1 - получение контекстно-зависимой справки;

Shift-F1 - выбор справки из списка доступных справочных сообщений;

Ctrl-F1 - получение справки о нужной стандартной процедуре, функции, о стандартной константе или переменной;

Alt-F1 - получение предыдущей справки.

По команде Shift-F1 на экране появится окно, содержащее упорядоченный по алфавиту список стандартных процедур, функций, типов, констант и переменных, для которых можно получить справочную информацию.

Эту же справку можно получить и по-другому. Напечатайте на экране имя процедуры (функции, типа и т.д.) или подведите курсор к имеющемуся в тексте стандартному имени и нажмите Ctrl-F1. Среда проанализирует ближайшее окружение курсора, выделит имя и даст нужную справку.

Во многих случаях справка содержит небольшой пример, иллюстрирующий соответствующие возможности Турбо Паскаля. Не торопитесь запоминать его или записывать на листе бумаги: его можно «вырезать» из справки и перенести в окно редактора. Для этого после вызова справки нажмите Alt-E, выберите в появившемся дополнительном меню продолжение Copy examples (копировать примеры) и нажмите Enter -текст примера скопируется во внутренний буфер редактора. Для извлечения пример из буфера, нажмите Esc, чтобы выйти из справочной службы, подвести курсор к свободной строке в окне редактора, нажмите Shift-Insert (копирование содержимого буфера в текст программы) и Ctrl-K H, чтобы убрать выделение скопированного текста цветом.

Проверка на ошибки - "компиляция".

Имея в окне редактора текст программы, можно проверить его на ошибки - такие как ошибки синтаксиса (забыли поставить ";"), ошибки в служебных словах (написали не begin, а bigin) и другие.

Для того, чтобы это сделать, необходимо нажать клавиши **Alt-F9**: нажать **Alt** и удерживая его нажать **F9**.

После этого начнется процесс проверки и если все написано без ошибок, то появится окно успешной компиляции, в котором будет мигать надпись

"Compile successfull: Press any key"

Если же возникли какие-либо ошибки, то процесс проверки остановится на первой из них, переведя курсор в строку с ошибкой и вверху или внизу редактора, появится красная строка с названием ошибки. Процесс, который происходит при этом, называется "компиляция".
Компиляция - это перевод программы с языка программирования на язык машинный, понятный компьютеру.
При нажатии сочетания клавиш **Alt-F9** и происходит попытка компиляции, то есть ваша программа переводиться с Паскаля на машинный язык. И если этот процесс не будет прерван ошибками в синтаксисе, структуре или какими-либо еще, то программа может быть запущена и выполнена компьютером.
После удачного процесса компиляции программа помещается в память компьютера и оттуда может начать свое выполнение.
Для того, чтобы выполнить ее, служит комбинация клавиш **Ctrl-F9**. При нажатии на эти клавиши происходит описанный выше процесс компиляции и сразу после этого происходит попытка запуска скомпилированной программы.
Опять же, если нет ошибок, произойдет запуск программы, и она начнет работу, иначе высветится сообщение об ошибке - в виде красной строки вверху с названием ошибки и курсор переведется в строку с ошибкой.

1.2 Отладка программ пользователя в Turbo Pascal

Turbo Pascal предлагает усовершенствованную среду, с автоматическим управлением проектом, модульной организацией программ, высокой скоростью компиляции. Но, даже используя все эти предоставленные средства, программы пользователя могут содержать ошибки, которые приводят к неправильной работе программы.
В помощь пользователю Turbo Pascal предоставляет средства, необходимые для отладки его программы, способствующие устранению всех ошибок в программе, ее тщательному тестированию и выполнению. Turbo Pascal позволяет легко определять местоположение ошибок во время компиляции и во время выполнения программы, а также позволяет включать или выключать автоматический контроль ошибок во время выполнения программы.
Особенно важно то, что Turbo Pascal имеет мощный и гибкий отладчик исходного уровня, который позволяет пользователю выполнять программу построчно, просматривать выражения и модифицировать переменные по мере необходимости. Отладчик встроен в интегрированную среду разработки (IDE) Turbo Pascal; пользователь может редактировать, компилировать и отлаживать программу, даже не выходя из Turbo Pascal. Для больших или сложных программ, которые требуют использования всего диапа-

зона средств отладки от машинного языка до вычисления выражений Паскаля, Turbo Pascal полностью поддерживает автономный отладчик фирмы Borland, Turbo Debugger.

Типы ошибок.

Существует три основных типа программных ошибок: ошибки времени компиляции, ошибки времени выполнения и логические ошибки.

Ошибки компиляции.

Ошибки компиляции или синтаксические ошибки встречаются, когда забывают объявить переменную, передают ошибочное количество параметров процедуры, при назначении действительного значения целочисленной переменной. Это означает, что записываются операторы, которые не согласуются с правилами Паскаля.

Turbo Pascal не закончит процесс компиляции программы пользователя (генерацию машинного кода), пока все синтаксические ошибки не будут удалены. Если Turbo Pascal обнаружит синтаксическую ошибку во время компиляции программы, он останавливает компиляцию, входит в исходный текст, указывает местоположение ошибки позиционированием курсора и выводит сообщение об ошибке в окно Edit. Как только пользователь исправит ошибку, он сможет начать процесс компиляции снова.

Ошибки времени выполнения.

Другой тип ошибок - ошибки времени выполнения программы или семантические ошибки. Они встречаются, когда пользователь компилирует синтаксически корректную программу, которая пытается сделать что-нибудь запрещенное во время ее выполнения, например, открывает несуществующий файл для ввода или производит деление на 0. В этом случае Turbo Pascal выводит на экран следующее сообщение об ошибке: Runtime error ## at seg:ofs (Ошибка выполнения # в сегменте: смещение) и останавливает выполнение программы пользователя.

При использовании интегрированной среды Turbo Pascal определяет местоположение ошибки выполнения автоматически, осуществляя переход в окно редактирования для соответствующего исходного файла.

Если пользователь выполняет программу в среде MS-DOS, он будет возвращаться в MS-DOS. Пользователь может загрузить модуль TURBO.EXE и использовать опции Search/Find error для локализации позиции ошибки в исходной.

Логические ошибки.

Программа пользователя может содержать и логические ошибки. Это означает, что программа делает то, что ей указали вместо того, что хотелось бы. Может отсутствовать инициализация переменной; могут оказаться ошибочными вычисления; рисунки, изображенные на экране, выглядят неправильно; программа может просто работать не так, как было задумано.

Такие ошибки находятся с большим трудом, и интегрированный отладчик поможет в этом случае наилучшим образом.

Интегрированный отладчик Turbo Pascal.

Некоторые ошибки времени выполнения (логические ошибки) незаметны и трудны для прослеживания. Другие ошибки могут скрываться за неуловимым взаимодействием разделов большой программы. В этих случаях необходимо интерактивное выполнение программы, во время которого производится наблюдение за значениями определенных переменных или выражений. Вам хотелось бы, чтобы Ваша программа останавливалась при достижении определенного места так, чтобы просмотреть, как она проработала этот кусок. Вам хотелось бы остановиться и изменить значения некоторых переменных во время выполнения программы, изменить определенный режим или проследить за реакцией программы. И вам хотелось бы сделать это в режиме, когда возможно быстрое редактирование, перекомпилирование и повторное выполнение программы.

Интегрированный отладчик Turbo Pascal имеет все описанные выше возможности и даже более того. Он представляет собой встроенную часть интегрированной усовершенствованной среды Turbo Pascal (IDE): для использования предлагаются две основные функции меню (Run, Debug), а также некоторые клавиши для команд отладчика.

Обзор возможностей отладчика:

-Трассировка. F7

Run/Trace Into Вы можете выполнить одну строку вашей программы, затем прерваться и посмотреть на результаты. При вызове процедуры или функции внутри программы, Вы можете задать режим выполнения вызова как одного шага или режим трассировки этой процедуры или функции строка за строкой.

Вы можете так же трассировать вывод программы строка за строкой. Вы можете так же установить, чтобы экран переключался по необходимости или использовать два монитора. Вы можете так же установить экран вывода в отдельном окне.

-Переход на курсор. F4

Run/Go to Cursor Вы можете передвинуть курсор на определенную строку в программе, а затем указать отладчику выполнить программу до достижения этой строки. Это позволяет обходить циклы или другие утомительные участки программы, это также позволяет перебираться в то место программы, откуда Вы хотите начать отладку.

-Прерывание.

С помощью команды Debug/Breakpoints Вы можете пометить строки в программе как точки прерывания. Когда в процессе выполнения Вашей программы достигается точка прерывания, выполнение программы приостанавливается и отображается исходный текст и курсор останавливается на

строке с точкой прерывания. Затем Вы можете проверить значения переменных, начать трассировку или выполнить программу до другой точки прерывания. Вы можете подключить условие к точке прерывания. Вы можете также прерваться в любой точке программы, нажав клавишу Ctrl-Break. Произойдет остановка на следующей строке исходной программы, как если бы в этой строке была установлена точка прерывания.

-Наблюдение.

Debug/Watches Пользователь имеет возможность задавать для просмотра в окне Watch некоторые объекты (переменные, структуры данных, выражения). Просматриваемые данные меняются, отражая текущие изменения в программе при пошаговом выполнении.

-Вычисление/модификация Ctrl-F4.

Debug/Evaluate/Modify Пользователь может вызвать окно Evaluate, что проверить значения переменных, структуру данных и выражения в интерактивном режиме. Используя окно Evaluate, Вы можете изменить значение любой переменной, включая строки, указатели, элементы массива и поля записей. Это обеспечивает простой механизм для проверки, как Ваш код реагирует на определенную установку значений или условий.

-Поиск.

Пользователь может быстро находить объявления процедур или функций, даже если программа разбита на несколько модулей (Search/Find Procedure). Во время трассировки Вы можете быстро вернуться обратно из вызовов процедур или функций и проверить параметры каждого вызова (Window/Call Stack).

Подготовка к использованию отладчика.

До начала отладки Вы должны понимать, что основным элементом выполнения в отладчике является строка, а не оператор. Более точно наименьшим элементом выполнения является строка. Если на одной строке находится несколько операторов, они будут выполняться вместе при нажатии F7. С другой стороны, если один оператор размещен на нескольких строках, то при нажатии F7 будет выполняться весь оператор. Все команды выполнения основываются на строках, включая пошаговую отладку и точки прерывания; строка, на которой находится выполнение, всегда отмечена курсором выполнения.

Прежде, чем начать отладку программы, Вы должны задать для компилятора Turbo Pascal инструкцию по генерации таблицы символов и таблицы номеров строк этой программы. Таблица символов представляет собой небольшую базу данных со всеми используемыми идентификаторами - константами, типами, переменными, процедурами и информацией о номерах строк. Директивы компилятора $D+ и $L+ делают это по умолчанию; они соответствуют элементам меню Options/Compiler/Debug Information и Options/Compiler/Local Symbols. Так же по умолчанию установлена опция

Options/Debugger/Integrated, которая генерирует отладочную информацию в выполнимом файле.
Директива {$D+} генерирует таблицу номеров строк, которая устанавливает соответствие между объектным кодом и исходным модулем. Директива {$L+} генерирует локальную отладочную информацию, а именно, строит список идентификаторов, локальных для каждой процедуры или функции, для того, чтобы отладчик мог хранить информацию о них в процессе отладки. Когда Вы используете директивы компилятора, разделяйте их запятыми и без пробелов, и ставя $ только перед первой директивой; например {$D+,L+}.
Примечание: Вы можете отключить эти переключатели для сохранения памяти или дискового пространства во время компиляции.
Когда Вы выполняете пошаговую отладку, Turbo Pascal будет иногда переключаться на экран пользователя, выполнять Ваш код, а затем возвращаться в интегрированную среду, ожидая следующей команды. Вы можете управлять переключением экрана с помощью установок Options/Debugger/Display Swapping, которые могут принимать 3 значения:

- Smart: Это режим по умолчанию. Среда IDE переключается на экран пользователя, когда программа обращается к видеопамяти или при вызове программы.
- Always: Переключение на экран пользователя происходит на каждом шаге.
- None: Переключение экранов не происходит. Интегрированная среда остается видимой все время. Если в программе предусматривается вывод на экран или требуется ввод информации, текст будет писаться на экране среды. Вы можете восстановить окна интегрированной среды, выбирая Ё/Refresh Display.

Начало сеанса отладки.

Наиболее быстрый способ начать отладку состоит в загрузке программы и выборе команды Run/Trace Into (F7). Программа будет компилироваться. Когда компиляция завершится, редактор отобразит на дисплей тело основной программы с индикацией строки выполнения на первом операторе begin. Пользователь может начать трассировку программы с этого места (нажать клавиши F7 или F8) или использовать другие методы которые приведены ниже.
Если пользователю необходимо начать отладку с определенного места программы, он может выполнить программу до этого места, а затем остановиться. Для этого, загрузите нужный раздел исходного модуля в редактор и передвиньте курсор на строку, где Вы желаете остановиться. Затем можно поступить двумя способами:

• Выбрать команду Run/Goto Cursor (или нажать клавишу F4), которая будет выполнять программу пользователя до достижения строки, помеченной курсором, а затем останавить работу программы.
• Задать на указанной строке точку прерывания (выбрать команду Debug/Toggle Breakpoint или нажать на Ctrl-F8), затем выполнить программу (выполнить команду Run/Run или нажать Ctrl-F9); остановка будет происходить каждый раз при достижении заданной строки. Вы можете задать несколько точек прерывания, в этом случае программа будет делать остановку всякий раз при достижении какой-либо из этих точек.

Рестарт сеанса отладки.

Если в процессе отладки программы возникает необходимость начать все сначала, то нужно выполнить команду Program Reset из меню Run. Система отладки повторно инициализируется, и команда следующего шага вернет вас к первой строке главной программы. При этом производится закрытие всех файлов, которые были открыты программой, очистка стека всех вложенных программ, которые вызывались программой, и освобождение всего использованного пространства кучи. Переменные программы, однако, не будут повторно инициализированы или подвержены модификации какого-нибудь другого вида. (Turbo Pascal никогда не инициализирует переменные автоматически). Однако, начальные значения типированных констант программы будут восстановлены.

Turbo Pascal также предлагает рестарт, если Вы производите какие-либо изменения в программе во время отладки. Например, если Вы изменяете часть программы, а затем выбираете любую команду выполнения (нажимаете клавиши F7, F8, F4, Ctrl-F9 и т.д.), Вы получите сообщение : Source modified, rebuild? (Y/N) (исходный модуль модифицирован, нужно повторить сборку? да/нет). Если Вы отвечаете Y, Turbo Pascal будет перекомпилировать программу и возобновит отладку программы с начала. Если Вы ответите N, Turbo Pascal предполагает, что Вы уверены в своих действиях, и продолжает сеанс отладки дальше. (Любые изменения в программе, которые Вы произвели, не будут влиять на ее выполнение до тех пор, пока Вы не перекомпилируете программу). Если Вы добавили или удалили строки программы, курсор выполнения не реагирует на эти изменения, и может оказаться, что будет выделяться ошибочная строка.

Окончание сеанса отладки.

В процессе отладки программы Turbo Pascal хранит трассу того, что Вы делаете и где находитесь в данный момент. Так как пользователь может в процессе отладки загружать и даже редактировать различные файлы, Turbo Pascal не интерпретирует загрузку другого файла в редактор, как конец сеанса отладки. Поэтому, если вы желаете выполнить или отладить другую программу, нужно выполнить команду Run/Program Reset (клавиша Ctrl - F2).

Трассировка программы.

Простейшая техника отладки - это пошаговая отладка, которая трассирует внутри процедур и функций.

Пошаговое выполнение программы.

Различие между командами Trace Into (F7) и Step Over (F8) в том, что при использовании F7 осуществляется трассировка внутри процедур и функций, в то время как использование F8 приведет к обходу вызовов подпрограмм. Эти команды имеют особое значение при выполнении оператора begin основной программы, если программа использует модули, имеющие раздел инициализации. В этом случае, использование F7 приведет к трассировке раздела инициализации каждого модуля, что позволяет увидеть, что инициализируется в каждом модуле. При использовании F8 эти разделы не будут трассироваться, и курсор выполнения переходит на следующую строку после begin.

Использование точек прерывания.

Точки прерывания являются важным инструментом отладки. Точка прерывания подобна знаку остановки, введенному в программу пользователя. Когда программа встречает такую точку, она останавливает свое выполнение и ожидает дальнейших отладочных инструкций.

Примечание: Вы можете иметь до 16 активных точек прерывания.

Заметим, что точки прерывания существуют только во время сеанса отладки; они не сохраняются в файле .EXE, если программа компилируется на диск. Чтобы задать точку прерывания, используйте обычные команды редактирования для перемещения курсора на каждую строку программы, где Вы хотите сделать паузу. Каждый раз выполняйте команду Debug/Toggle Breakpoint (Ctrl-F8). Когда строка отмечается как точка прерывания, она высвечивается. Это не должна быть пустая строка, комментарии, директивы компиляции; объявления констант, типов, меток, переменных; заголовком программы, модуля, процедуры или функции. Как только Вы задали точки прерывания, выполняйте программу с помощью команды Run/Run (клавиша Ctrl-F9). Сначала программа будет выполняться нормально. Когда встретится точка прерывания, программа остановится. Соответствующий исходный файл (основная программа, модуль или включенный файл) загружается в окно Edit, которое визуализируется на экране и курсор выполнения помещается на строку с точкой прерывания.

Заметим, что точка прерывания не высвечивается, когда на ней находится курсор выполнения. Если какие-либо переменные или выражения были добавлены в окно Watch, то они также выводятся на дисплей со своими текущими значениями.

Затем, пользователь может использовать любой режим отладки.

- Вы можете осуществлять пошаговое выполнение программы, используя команду Run / Trace Into, Step Over или Go to Cursor (F7, F8 или F4). Вы можете проверить или изменить значения переменных.
- Вы можете добавить или удалить выражения из окна Watch.
- Можно назначить или удалить точки прерывания.
- Можно просмотреть выходные результаты программы, используя команду Windows/User Screen (Alt-F5).
- Вы можете перезапустить программу сначала (Run/Program Reset и, затем, команду пошагового выполнения).
- Можно продолжить выполнение до следующей точки прерывания (или до конца программы), выполнив команду Run/Run (Ctrl-F9).

Для удаления точки прерывания из строки переместите курсор на данную строку и, выполнив команду Debug/Toggle Breakpoint (или нажмите Ctrl-F8) еще раз. Эта команда включает или отключает точку прерывания в строке; если она используется для строки с точкой прерывания, то строка становится нормальной.

Использование Ctrl-Break.

Кроме назначения точек прерывания, пользователь может сделать немедленную остановку во время выполнения программы, используя клавишу Ctrl-Break. Это означает, что можно прервать работу программы в любое время. Когда Вы нажимаете на клавишу Ctrl-Break, выполнение программы прекращается. Вы возвращаетесь в редактор, курсор выполнения расположен на следующей строке программы, и программа готова к дальнейшему пошаговому выполнению.

Фактически, отладчик автоматически подключает DOS, BIOS и другие сервисные функции. Он знает, является ли текущий выполняющийся код программой DOS, программой BIOS или программой пользователя. Когда Вы нажимаете на клавишу Ctrl-Break, отладчик ждет, пока программа выполняется сама. Затем он делает пошаговое выполнение инструкций машинного уровня, пока следующая инструкция не будет в начале строки исходного кода на Паскале. С этого момента отладчик прекращает работу, перемещает курсор выполнения на эту строку и предлагает Вам нажать на клавишу ESC.

Примечание: Если пользователь нажимает на клавишу Ctrl-Break второй раз еще до того, как отладчик находит и отображает следующую строку исходного кода для выполнения, то отладчик завершает работу и не пытается найти строку исходного кода. В этом случае, процедуры выхода не выполняются, что означает, что файлы, видеорежим и распределение памяти DOS могут быть не полностью очищены.

Просмотр значений.

Выполнение программы предоставляет много информации, но не в том объеме, как хотелось бы. Может возникнуть необходимость просмотреть

за тем, как изменяются значения переменных во время выполнения программы. Отладчик дает возможность пользователю задать объекты для просмотра во время выполнения программы. Объектами просмотра являются переменные, структуры данных и выражения, расположенные в окне Watch, где отображаются их текущие значения, обновляемые по мере выполнения каждой строки программы. Чтобы установить объекты наблюдения, надо передвигать курсор к каждому идентификатору и выполнять команду Debug/Watch/Add Watch (Ctrl -F7) для добавления каждого выражения в окно Watch.

Вопросы для самопроверки:

1. Как создать, сохранить и открыть файл в Turbo Pascal? Какое расширение имеет файл в Turbo Pascal?
2. Что такое компиляция программы и как осуществить эту процедуру в Turbo Pascal? Как запустить программу в Turbo Pascal?
3. Как можно выделить, скопировать, вырезать, переместить фрагмент программы в Turbo Pascal?
4. Какие основные функциональные клавиши используются в Turbo Pascal?
5. Как создать файл в TP 7.0 и скопировать в него фрагмент программы из другого файла?

Глава 2. Основные элементы языка

2.1 Алфавит языка и структура программы

Алфавит

Алфавит языка Турбо-Паскаль (набор используемых символов) включает буквы латинского алфавита: от a до z и от A до Z, арабские цифры: от 0 до 9, специальные символы: _ + - * / = , . : ; < > ( ) [ ] { } ^ @ $ #, пробел "_" и управляющие символы с кодами от #0 до #31.

Используются также служебные (зарезервированные) слова, например:
absolute, and, array, begin, case, const, constructor, destructor, div, do, downto, else, end, external, file, for, forward, function, goto, if, implementation, in, inline, interface, interrupt, label, mod, nil, not, object, of, or, packed, procedure, program, record, re-peat, set, shl, shr, string, then, to, type, unit, until, uses, var, vertual, while, with, xor и другие.

Общая структура программы в Turbo Pascal.

{Комментарий на русском языке}
Program Name; {Это-заголовок программы - не более 256 лат. букв, цифр}
{Раздел описаний}

USES - Список используемых библиотек или модулей (например, CRT, GRAPH);
LABEL - Список меток в основном блоке программы (например, m1,m2);
CONST - Определение констант программы;
TYPE - Описание типов;
VAR - Определение глобальных переменных программы;
ОПРЕДЕЛЕНИЕ ПРОЦЕДУР (заголовки и, возможно, тела процедур);
ОПРЕДЕЛЕНИЕ ФУНКЦИЙ (заголовки и, возможно, тела функций);
BEGIN {Это-символ начала программы}
Основной блок (тело) программы (раздел исполняемых операторов);
END. {Это-символ окончания программы}

В Паскале существует понятие **переменной**. Переменная служит для хранения какого-либо значения и имеет собственное имя. Это имя ей необходимо назначить - оно может состоять только из:

1. латинских букв **(A..Z)**;
2. цифр **(0..9)** (но не может начинаться с цифры);
3. символов подчеркивая **"_"** ;

И не может содержать:

1. Русских букв;
2. Любых знаков препинания, таких как точка, запятая, восклицательный знак;
3. Специальных символов, которые находятся над цифрами на клавиатуре. Это "~", "#", "$", "%" и другие.

Примеры имен переменных:
primer1; _primer; Primer; _pr_; my_name_is_Edik;

Замечание: имя переменной может быть любой длины, не превышающей 250 символов. Также не учитывается регистр букв, то есть переменные с именами **Primer** и **pRiMeR** будут рассматриваться как одна.

Тип переменной определяет, что с этой переменной можно сделать, и из чего она состоит (что в ней содержится). Для каждой переменной определяется ее тип.

Нельзя сложить число со строкой, так как цифры с символами складывать невозможно. Поэтому если определить тип одной переменной как числовой, а тип второй переменной, как строковый. Тогда Паскаль будет знать, что с какой переменной можно сделать, и при произведении каких-либо действий с переменными определять их тип, после чего, либо производить действия, либо нет, если их произвести невозможно.

Для типа переменной существуют разные обозначения. Например, тип "число" обозначается как **Integer**, что по-английски означает "число". Зная, что переменная типа **Integer,** можно ее складывать с другими, вычитать или умножать.

Структура программы:

1. **Program (имя);** - это заголовок программы. Совсем необязательный, программисты просто пишут его для того, чтобы как-то озаглавить программу. Его хорошо использовать для маркировки, т.е. чтобы по нему определять, что это за программа. Всегда находится в первой строчке программы, если его разместить где-нибудь в другом месте, то это вызовет ошибку.
2. **var** - это раздел описания переменных. Выше было сказано, что надо вводить необходимые переменные и придавать им определенный тип. Вот здесь это и реализуется. После служебного слова **var**, сообщающего Паскалю, что начался раздел объявления переменных, надо расставить все переменные, и через двоеточие указать их тип. Например, числа указываются словом **Integer**. Т.е., **A,B,C: Integer;**
3. **begin** - это служебное слово означает, что начался раздел действий. Именно после него программа начинает свое выполнение. По-английски "begin" - значит "начало". Когда Паскаль встречает это слово, он начинает выполнение программы. У слова **begin** есть завершающая пара - **end**. Его можно увидеть в самом конце программы. Это слово совершенно противоположное по значению - то есть оно означает, что выполнение программы закончилось. Именно пара **begin - end.** и есть главной в программе, между ней находятся все действия.
4. **Выражения -** то, что необходимо сделать.

Синтаксис выражений.

Самое главное правило - после каждой команды или выражения ставится точка с запятой - **";"** Каждая строка имеет свое завершение этим знаком. Исключение составляют только слова **var** и **begin** - это служебные слова, которые не являются процедурами или функциями. Они определяют начало какого-либо раздела программы, в частности, **var** - начало раздела объявления переменных, а **begin** - раздела выполнения. Кроме этих служебных слов есть и другие, например, начало раздела констант. Завершение имеет только раздел **begin** словом **end**, после которого ставиться точка. Точка означает конец программы, ее завершение и полную остановку выполнения.
Отсюда имеет несколько правил синтаксиса:

1. После выражений ставится **";"**
2. Служебные слова, означающие начало раздела программы, не оканчиваются знаком **";"**

2.2 Арифметические операции и стандартные функции

Арифметическим называется выражение, составленное из операндов – величин, над которыми производится операция, скобок и знаков операций. В результате вычисления выражения получается значение определенного типа. Порядок вычисления выражения определяется скобками и старшин-

ством операций. Они делятся на арифметические, отношения, логические и другие. Операции могут быть унарными и бинарными.

Арифметические операции

Операция	Действие	Тип операндов	Тип результата
Бинарные			
+	Сложение	Целый, вещественный	Целый, вещественный
-	Вычитание	Целый, вещественный	Целый, вещественный
*	Умножение	Целый, вещественный	Целый, вещественный
/	Деление	Целый, вещественный	Вещественный
DIV	Целочисленное деление	Целый	Целый
MOD	Остаток от деления	Целый	Целый
Унарные			
+	Сохранение знака	Целый, вещественный	Целый, вещественный
-	Отрицание знака	Целый, вещественный	Целый, вещественный

Операции отношения

Операции отношения выполняют сравнение двух операндов и определяют, истинно значение или ложно. Сравниваемые величины могут принадлежать к любому типу данных, и результат всегда имеет логический тип, принимая одно значение из двух: истина или ложь.

Операция	Название	Выражение
=	Равно	A=B
<>	Неравно	A<>B
>	Больше	A>B
<	Меньше	A<B
>=	Больше или равно	A>=B
<=	Меньше или равно	A<=B

Стандартные математические функции

Обращение	Тип аргумента	Тип результата	Функция
Abs(x)	Целый, вещественный	Целый, вещественный	Модуль аргумента

Arctan(x)	Целый, вещест-венный	Вещественный	Арктангенс
Cos(x)	Целый, вещест-венный	Вещественный	Косинус
Exp(x)	Целый, вещест-венный	Вещественный	e^x - экпонента
Frac(x)	Целый, вещест-венный	Вещественный	Дробная часть x
Int(x)	Целый, вещест-венный	Вещественный	Целая часть x
Ln(x)	Целый, вещест-венный	Вещественный	Натуральный лога-рифм
Random		Вещественный	Псевдослучайное число [0,1]
Random(x)	Целый	Целый	Псевдослучайное число [0,x]
Round(x)	Вещественный	Целый	Округление до бли-жайшего целого
Sin(x)	Целый, вещест-венный	Вещественный	Синус
Sqr(x)	Целый, вещест-венный	Вещественный	Квадрат x
Sqrt(x)	Целый, вещест-венный	Вещественный	Корень квадратный из x
Trunc(x)	Вещественный	Целый	Ближайшее целое, не превышающее x по модулю

Логические операции

Логические выражения в результате вычисления принимают логические значения True и False. Операндами это выражения могут быть логические константы, переменные, отношения. Идентификатор логического типа в Паскале: boolean.

В Паскале имеется 4 логические операции: отрицание -NOT, логическое умножение -AND, логическое сложение - OR, исключающее «или» -XOR . Использованы обозначения: T – true, F – false.

A	B	Not A	A and B	A or B	A xor B
T	T	F	T	T	F
T	F	F	F	T	T
F	F	T	F	F	F
F	T	T	F	T	T

Приоритеты операций: not, and, or, xor. Операции отношения (= , <> …) имеют более высокий приоритет, чем логические операции, поэтому их следует заключать в скобки при использовании по отношению к ним логических операций.

Приоритет операций (в порядке убывания):

- вычисление функции;
- унарный минус, not;
- умножение, деление, div, mod, and;
- сложение, вычитание, or, xor;
- операции отношения

Вопросы для самопроверки:

1. Какова общая структура программы в Turbo Pascal?
2. Что такое операторы присваивания, ввода и вывода информации?
3. Какие служебные (зарезервированные) слова в Turbo Pascal Вы знаете?
4. Что является заголовком, символом начала и символом конца программы в Turbo Pascal?
5. Как производится запись математических формул в Turbo Pascal?
6. Какие производятся операции с действительными числами и какие Вы знаете встроенные функции в Turbo Pascal?

Глава 3. Простые типы данных

При решении задач выполняется обработка информации различного свойства, например дробные и целые числа, слова, строки и т.д. Для описания множества допустимых значений величины и совокупности операций, в которых участвует данная величина, используется указание ее типа данных.

Тип данных – это множество величин, объединенных определенной совокупностью допустимых операций. Каждый тип имеет свой диапазон значений и специальное зарезервированное слово для описания. Все типы данных можно разделить на две группы: скалярные (простые) и структурированные (составные). Простые типы данных также делятся на стандартные и пользовательские. Стандартные – предлагаются разработчиками Турбо Паскаля, а пользовательские разрабатывают сами программисты.

1. Простые типы
   Порядковые типы
      Целые типы
      Логический тип
      Символьный тип
      Перечисляемый тип
      Интервальный тип

Вещественные типы

2. Структурированные типы

Строковый тип

Регулярный тип

Множественный тип

Файловый тип

К простым типам относятся порядковые и вещественные типы. Порядковые типы отличаются тем, что каждый из них имеет конечное число возможных значений. Эти значения можно определенным образом упорядочить (отсюда - название типов) и, следовательно, с каждым из них можно сопоставить некоторое целое число - порядковый номер значения.

Вещественные типы, строго говоря, тоже имеют конечное число значений, которое определяется форматом внутреннего представления вещественного числа. Однако количество возможных значений вещественных типов настолько велико, что сопоставить с каждым из них целое число (его номер) не представляется возможным.

К порядковым типам относятся целые, логический, символьный, перечисляемый и тип-диапазон. К любому из них применима функция ORD(X), которая возвращает порядковый номер значения выражения X. Для целых типов функция ORD(X) возвращает само значение X, т.е. ORD(X) = X для X, принадлежащего любому шелому типу. Применение ORD(X) к логическому, символьному и перечисляемому типам дает положительное целое число в диапазоне от 0 до 1 (логический тип), от 0 до 155 (символьный), от 0 до 65535 (перечисляемый). Тип-диапазон сохраняет все свойства базового порядкового типа, поэтому результат применения к нему функции ORD(X) зависит от свойств этого типа.

К порядковым типам можно также применять функции:

PRED (X) - возвращает предыдущее значение порядкового типа (значение, которое соответствует порядковому номеру ORD(X)- 1), т.е.

ORD(PRED(X)) = ORD(X) - 1;

SUCC (X) - возвращает следующее значение порядкового типа, которое соответствует порядковому номеру ORD(X) +1, т.е.

ORD(SUCC(X)) = ORD(X) + 1.

Например, если в программе определена переменная

```
var
c : Char;
begin
c := '5' ;
end.
```

то функция PRED(C) вернет значение '4', а функция SUCC(C) - значение '6'.

Если представить себе любой порядковый тип как упорядоченное множество значений, возрастающих слева направо и занимающих на числовой оси некоторый отрезок, то функция PRED(X) не определена для левого, а SUCC(X) - для правого конца этого отрезка.

Целочисленные типы данных

Тип	Диапазон	Требуемая память (байт)
Byte	0…255	1
Shorint	-128 …127	1
Integer	-32768 … 32767	2
Word	0 … 65535	2
Longint	-2147483648 …2147483647	4

Значения целых типов могут изображаться в программе 2 способами: в десятичном виде и в шестнадцатеричном. Если число представлено в шестнадцатеричной системе, перед ним без пробела ставится знак $, а цифры старше 9 обозначаются латинскими буквами от A до F. Диапазон изменений таких чисел от $0000 до $FFFF .

Вещественные типы данных

Вещественные (действительные) типы данных представляют собой значения, которые используются в арифметических выражениях и могут быть представлены двумя способами: с фиксированной и с плавающей точкой.

Тип	Диапазон	Мантисса	Требуемая память (байт)
Real	2.9*10E-39 … 1.7*10E38	11 – 12	6
Single	1.5*10E-45 … 3.4*10E38	7 – 8	4
Double	5.0*10E-324 … 1.7*10E308	15 – 16	8
Extended	1.9*10E-4951 … 1.1*10E4932	19 – 20	10
Comp	-2E+63+1 … 2E+63-1	10 – 20	8

Примечание. Все вещественные типы, кроме Real, могут быть использованы в программе при наличии в ПК математического сопроцессора Intel 8087/80287.

Действительные числа с фиксированной точкой записываются по обычным правилам арифметики, только целая часть от дробной отделяется точкой. Если точка отсутствует, число считается целым. Перед числом может стоять знак «+» или «-». Если знака нет, то число считается положительным.

Числа в форме с плавающей точкой представляются в экспоненциальном виде: mE+p, где m – мантисса (целое или дробное число), E означает 10 в степени, p – порядок (целое число).

Например, $5.18E+2 = 5.18 * 10^2 = 518$

$10E-03 = 10 * 10^{-3} = 0.01$

Символьный тип

Литерный (символьный) тип char определяется множеством значений кодовой таблицы ПК. Каждому символу приписывается целое число в диапазоне от 0 до 255. Для кодировки используется код ASCII. Например код символа 'A' при русской раскладке клавиатуры будет равен 192.

Для размещения в памяти переменной литерного типа нужен 1 байт.

Логический тип

Логический (булевский) тип boolean определяется двумя значениями: true (истина) и false (ложь). Он применяется в логических выражениях и выражениях отношения. Для размещения в памяти - 1 байт.

Перечисляемый тип.

Перечисляемый тип задается перечислением тех значений, которые он может получать. Каждое значение именуется некоторым идентификатором и располагается в списке, обрамленном круглыми скобками, например:

```
type
colors =(red, white, blue);
```

Применение перечисляемых типов делает программы нагляднее. Например,

```
type
TypeMonth=(jan,feb,mar,may,jun,jul,aug,sep,oct,nov,dec);
var
month: TypeMonth;
begin
.......
if month = aug then WriteLn('Хорошо бы поехать к морю!');
.......
end.
```

Соответствие между значениями перечисляемого типа и порядковыми номерами этих значений устанавливается порядком перечисления: первое значение в списке получает порядковый номер 0, второе - 1 и т.д. Максимальная мощность перечисляемого типа составляет 65536 значений, поэтому фактически перечисляемый тип задает некоторое подмножество це-

лого типа WORD и может рассматриваться как компактное объявление сразу группы целочисленных констант со значениями 0, 1 и т.д.
Использование перечисляемых типов повышает надежность программ благодаря возможности контроля тех значений, которые получают соответствующие переменные. Пусть, например, заданы такие перечисляемые типы:

```
type
colors = (black, red, white);
ordenal= (one, two, three);
days = (monday, tuesday, Wednesday);
```

С точки зрения мощности и внутреннего представления все три типа эквивалентны:

```
ord(black)=0, ..., ord(white)=2,
ord(one)=0, ...ord(three)=2,
ord(monday)=0, ...ord(Wednesday)=2.
```

Однако, если определены переменные

```
var
col :colors; num :ordenal;
day : days;
```

то допустимы операторы

```
col := black;
num := succ(two);
day := pred(tuesday);
```

но недопустимы

```
col := one;
day := black;
```

Как уже упоминалось, между значениями перечисляемого типа и множеством целых чисел существует однозначное соответствие, задаваемое функцией ORD(X). В Турбо Паскале допускается и обратное преобразование: любое выражение типа WORD можно преобразовать в значение перечисляемого типа, если только значение целочисленного выражения не превышает значения перечисляемого типа. Такое преобразование достигается применением автоматически объявляемой функции с именем перечисляемого типа. Например, для рассмотренного выше объявления типов эквивалентны следующие присваивания:

```
col := one;
col := colors(0);
```

Разумеется, присваивание

```
col := 0;
```

будет недопустимым.
Переменные любого перечисляемого типа можно объявлять без предварительного описания этого типа, например:

```
var
```

```
col: (black, white, green);
```

Тип-диапазон.

Тип-диапазон есть подмножество своего базового типа, в качестве которого может выступать любой порядковый тип, кроме типа-диапазона. Тип-диапазон задается границами своих значений внутри базового типа:

<мин.знач.>..<макс.знач.>

Здесь <мин.знач. > - минимальное значение типа-диапазона;
<макс.знач.> - максимальное его значение.

Например:

```
type
digit = '0'..'9';
dig2= 48..57;
```

Тип-диапазон необязательно описывать в разделе TYPE, а можно указывать непосредственно при объявлении переменной, например:

```
var
date : 1..31;
month: 1. .12;
Ichr : 'A'..'Z';.
```

При определении типа-диапазона нужно руководствоваться следующими правилами:

- два символа «..» рассматриваются как один символ, поэтому между ними недопустимы пробелы;
- левая граница диапазона не должна превышать его правую границу. Тип-диапазон наследует все свойства своего базового типа, но с ограничениями, связанными с его меньшей мощностью. В частности, если определена переменная

```
type
days = (mo,tu,we,th,fr,sa,su);
WeekEnd = sa .. su;
var
w : WeekEnd;
begin
.....
w := sa;
.....
end;
```

то ORD(W) вернет значение 5 , в то время как PRED(W) приведет к ошибке.

В стандартную библиотеку Турбо Паскаля включены две функции, поддерживающие работу с типами-диапазонами:

HIGH(X) - возвращает максимальное значение типа-диапазона, к которому принадлежит переменная X;

LOW(X) -возвращает минимальное значение типа-диапазона.

Вопросы для самопроверки:

1. Как и когда в Turbo Pascal используются данные действительного типа?
2. Как и в каких случаях в Turbo Pascal используются данные целого типа? Какие производятся операции над целыми числами?
3. Что такое число с плавающей и с фиксированной точкой? Что такое форматы чисел?
4. Что такое данные логического типа и как они используются?
5. Какие существуют операции отношений над переменными в Turbo Pascal?
6. Какие существуют операции над данными логического типа?
7. Что такое данные символьного типа?
8. Что такое данные строкового типа?

Глава 4. Операторы языка: простые, структурированные

Оператором называется предложение языка программирования, задающее полное описание некоторого действия, которое необходимо выполнить. Основная часть программы на языке Турбо Паскаль представляет собой последовательность операторов. Разделителем операторов служит точка с запятой. Операторы, не содержащие других операторов, называются простыми. К ним относятся операторы присваивания, безусловного перехода, вызова процедуры, пустой. Структурные операторы представляют собой конструкции, построенные из других операторов по строго определенным правилам. Эти операторы можно разделить на три группы: составные, условные и повтора.

4.1 Простые операторы

Пустой оператор

Пустой оператор не содержит никаких символов и не выполняет никаких действий. Используется для организации перехода к концу блока в случаях, если необходимо пропустить несколько операторов, но не выходить из блока.

Оператор присваивания.

Присваивание в Паскале обозначается знаком ":="; (двоеточие и равно). То есть, если мы хотим присвоить какой-либо переменной значение, то мы пишет эту переменную, ставим ":=" и пишем новое ее значение. Примеры присваиваний:

```
C := 15;
A := 15 + 3;
```

A := A + C + 3 - 12;

Оператор вызова процедуры

Оператор вызова процедуры служит для активизации стандартной процедуры или процедуры, определенной пользователем. Стандартные процедуры находятся в файлах, подключаемых модулем и для их использования достаточно указать имя процедуры, и если необходимо дополнительные параметры. Для того, чтобы вызвать свою процедуру, ее для этого надо описать перед началом программы (begin), а затем уже использовать.

Оператор безусловного перехода

Оператор безусловного перехода (go to) означает «перейти к» и применяется в случаях, когда после выполнения некоторого оператора надо выполнить не следующий по порядку, а какой-либо другой, отмеченный меткой, оператор. Общий вид: go to <метка>.

Метка объявляется в разделе описания меток и состоит из имени и следующего за ним двоеточия. Имя метки может содержать цифровые и буквенные символы, максимальная длина имени ограничена 127 знаками. Раздел описания меток начинается зарезервированным словом Label, за которым следует имя метки.

Пример.

```
Program primer;
Label 999, metka;
Begin
….
Go to 999;
…
999: write (‘ Имя’);
…
end.
```

4.2 Структурные операторы

Структурные операторы представляют собой конструкции, построенные из других операторов по строгим правилам. Применение структурных операторов в вашей программе очень часто просто незаменимо, потому что они позволяют программисту сделать его программу зависимой от каких-либо условий, например введенных пользователем. К тому же, применяя операторы повтора, вы получаете возможность обрабатывать большие объемы данных за сравнительно малый отрезок времени.

Составной оператор

Это оператор представляет собой совокупность произвольного числа операторов, отделенных друг от друга точкой с запятой, и ограниченную опе-

раторными скобками begin и end. Он воспринимается как единое целое и может находиться в любом месте программы, где возможно наличие оператора.

Условные операторы

Условные операторы предназначены для выбора к исполнению одного из возможных действий, в зависимости от некоторого условия (при этом одно из действий может отсутствовать). Для программирования ветвящихся алгоритмов в Турбо Паскале есть специальные операторы. Одним из них является условный оператор If. Это одно из самых популярных средств, изменяющих порядок выполнения операторов программы.

Он может принимать одну из форм:

```
If <условие> then <оператор1>
               else<оператор2>;
```

или

```
If <условие> then <оператор>;
```

Оператор выполняется следующим образом. Сначала вычисляется выражение, записанное в условии. В результате его вычисление получается значение логического (булевского) типа. Если это значение – «истина», то выполняется оператор1, указанный после слова then. Если же в результате имеем «ложь», то выполняется оператор2. В случае, если вместо оператора1 или оператора2 следует серия операторов, то эту серию операторов необходимо заключить в операторные скобки begin…end.

Обратить внимание, что перед словом else точка с запятой не ставится.

Если оператор If обеспечивает выбор из двух альтернатив, то существует оператор, который позволяет сделать выбор из произвольного числа вариантов. Это оператор выбора Case. Он организует переход на один из нескольких вариантов действий в зависимости от значения выражения, называемого селектором.

```
Общий вид: Case k of
                <const1>: <оператор1>;
                <const2>: <оператор2>;
                 …..
                <constN>: <операторN>
                else <операторN+1>
                end;
```

Здесь k – выражение-селектор, которое может иметь только простой порядковый тип (целый, символьный, логический). <const1>, …<constN> - константы того же типа, что и селектор.

Оператор Case работает следующим образом. Сначала вычисляется значение выражения-селектора, затем обеспечивается реализация того оператора, константа выбора которого равна текущему значению селектора. Если ни одна из констант не равна значению селектора, то выполняется опера-

тор, стоящий за словом else. Если же это слово отсутствует, то активизируется оператор, находящийся за границей Case, т.е. после слова end.

При использовании оператора Case должны выполняться следующие правила:

1. Выражение-селектор может иметь только простой порядковый тип (целый, символьный, логический).
2. Все константы, которые предшествуют операторам альтернатив, должны иметь тот же тип, что и селектор.
3. Все константы в альтернативах должны быть уникальны в пределах оператора выбора.

Формы записи оператора:

Селектор интервального типа:

```
Case I of
1..10    : writeln('число в диапазоне 1-10');
11.. 20 : writeln('число в диапазоне 11-20');
else writeln('число вне пределов нужных диапазонов')
end;
```

Селектор целого типа:

```
Case I of
1 : y:= I+10;
2 : y:= I+20;
3:  y:= I +30;
end;
```

Операторы повтора (цикла)

Если в программе возникает необходимость неоднократного выполнения некоторых операторов, то для этого используются операторы повтора (цикла). В языке Паскаль различают три вида операторов цикла: цикл с предусловием (while), цикл с постусловием (repeat) и цикл с параметром (for).

Если число требуемых повторений заранее известно, то используется оператор, называемый оператором цикла с параметром.

Оператор цикла с параметром имеет два варианта записи:

1) for <имя переменной> := <начальное значение> to <конечное значение> do
<тело цикла>
2) for <имя переменной> := <начальное значение> downto <конечное значение> do
<тело цикла>

Имя переменной – параметр цикла, простая переменная целого типа; <тело цикла> - операторы или оператор. Цикл повторяется до тех пор пока значение параметра лежит в интервале между начальным и конечным значе-

ниями. В первом варианте при каждом повторении цикла значение параметра увеличивается на 1, во втором - уменьшается на 1.

При первом обращении к оператору for вначале определяются начальное и конечное значения, и присваивается параметру цикла начальное значение. После этого циклически повторяются следующие действия.

1. Проверяется условие параметр цикла <= конечному значению.
2. Если условие выполнено, то оператор продолжает работу (выполняется оператор в теле цикла), если условие не выполнено, то оператор завершает работу и управление в программе передается на оператор, следующий за циклом.
3. Значение параметра изменяется (увеличивается на 1 или уменьшается на 1).

Если в теле цикла располагается более одного оператора, то они заключаются в операторные скобки begin … end;

Если число повторений заранее неизвестно, а задано лишь условие его повторения (или окончания), то используются операторы while и repeat. Оператор While часто называют оператором <u>цикла с предусловием</u>. Так как проверка условия выполнения цикла производится в самом начале оператора.

Общий вид: While <условие продолжения повторений> do
<тело цикла>;

Тело цикла – простой или составной оператор или операторы. Если операторов в теле цикла несколько, то тело цикла заключается в операторные скобки begin…end.

Перед каждым выполнением тела цикла вычисляется значение выражения условия. Если результат – «истина», тело цикла выполняется и снова вычисляется выражение условия. Если результат–«ложь», происходят выход из цикла и переход к первому после while оператору.

Оператор цикла repeat аналогичен оператору while, но отличается от него, во-первых, тем, что условие проверяется после очередного выполнения операторов тела цикла и таким образом гарантируется хотя бы однократное выполнение цикла. Во-вторых, тем, что критерием прекращения цикла является равенство выражения константе true . За это данный оператор часто называют <u>циклом с постусловием</u>, так как он прекращает выполняться, как только условие, записанное после слова until, выполнится. Оператор цикла repeat состоит из заголовка, тела и условия окончания.

Общий вид: Repeat
<оператор>
. . . . .
<оператор>
until <условие окончания цикла>

Вначале выполняется тело цикла, затем проверяется условие выхода из цикла. В любом случае этот цикл выполняется хотя бы один раз. Если условие не выполняется, т.е. результатом выражения является False, то цикл активизируется еще раз. Если условие выполнено, то происходит выход из цикла. Использования операторных скобок, в случае, если тело цикла состоит из нескольких операторов, не требуется.

Вопросы для самопроверки:

1. Какие управляющие конструкции в Turbo Pascal Вы знаете?
2. Что такое условный оператор и как он применяется?
3. Что такое оператор перехода и как он применяется?
4. Что такое операторы цикла?
5. Когда и как применяется оператор цикла for...to...do?
6. Когда и как применяется оператор цикла while...do?
7. Когда и как применяется оператор цикла REPEAT...UNTIL?

Глава 5. Составные, или структурированные, типы данных

Все простые типы данных, рассматриваемые ранее, имеют два характерных свойства: неделимость и упорядоченность их значений. Составные, или структурированные, типы данных задают множество сложных значений с одним общим именем. Существует несколько методов структурирования, каждый из которых отличается способом обращения к отдельным компонентам.

5.1 Массивы

Одномерные массивы.

С понятием «массив» приходится встречаться при решении научно-технических, экономических задач обработки большого количества однотипных значений.

Таким образом, массив – это упорядоченная последовательность данных, состоящая из фиксированного числа элементов, имеющих один и тот же тип, и обозначаемая одним именем.

Название регулярный тип массивы получили за то, что в них объединены однородные элементы, упорядоченные (урегулированные) по индексам, определяющим положение каждого элементы в массиве.

Массиву присваивается имя, посредством которого можно ссылаться на него, как на единое целое. Элементы, образующие массив, упорядочены так, что каждому элементу соответствует совокупность номеров (индексов), определяющих его место в общей последовательности. Индексы

представляют собой выражения простого типа. Доступ к каждому отдельному элементу осуществляется обращением к имени массива с указанием индекса нужного элемента: <имя массива>[<индекс>].
Описание массива определяет его имя, размер массива и тип данных. Общий вид описания массива:
Type <имя нового типа данных>=array[<тип индекса>] of <тип компонентов>;
Далее, в перечне переменных указывается имя массива, и через двоеточие указывается имя нового типа данных. Массив может быть описан и без представления типа в разделе описания типов данных:
Var <имя массива>: array [<тип индекса>] of <тип компонентов>;
Чаще всего в качестве типа индекса используется интервальный целый тип.
Одномерные массивы
Линейный (одномерный) массив – массив, у которого в описании задан только один индекс, если два индекса – то это двумерный массив и т.д. Одномерные массивы часто называют векторами, т.е. они представляют собой конечную последовательность пронумерованных элементов.
Присваивание начальных значений (заполнение массива) заключается в присваивании каждому элементу массива некоторого значения, заданного типа. Наиболее эффективно эта операция осуществляется при помощи оператора for. Ввод данных может осуществляться : с клавиатуры, при помощи различных формул, в том числе и датчика случайных чисел.
Индексированные элементы массива называются индексированными переменными и могут быть использованы так же, как и простые переменные. Например, они могут находиться в выражениях в качестве операндов, им можно присваивать любые значения, соответствующие их типу и т.д.
Алгоритм решения задач с использованием массивов:

- Описание массива
- Заполнение массива
- Вывод (распечатка) массива
- Выполнение условий задачи
- Вывод результата

Двумерные массивы

Двумерный массив – структура данных, хранящая прямоугольную матрицу. В матрице каждый элемент определяется номером строки и номером столбца, на пересечении которых он расположен. В Паскале двумерный массив представляется массивом, элементами которого являются одномерные массивы. Два следующих описания двумерных массивов тождественны:
Var a: array [1..10] of array [1.. 20] of real;
Var a: array [1..10, 1..20] of real;

Чаще всего при описании двумерного массива используют второй способ. Доступ к каждому отдельному элементу осуществляется обращением к имени массива с указанием индексов (первый индекс – номер строки, второй индекс – номер столбца). Все действия над элементами двумерного массива идентичны действиям над элементами линейного массива. Только для инициализации двумерного массива используется вложенный цикл for.

Например,
```
For i:= 1 to 10 do
    For j:= 1 to 20 do
    A[i, j] := 0;
```

При организации вложенных (сложных) циклов необходимо учитывать:

- Все правила, присущие простому циклу, должны соблюдаться
- Имена параметров для циклов, вложенных один в другой, должны быть различными
- Внутренний цикл должен полностью входить в тело внешнего цикла. Пересечение циклов недопустимо

5.2 Символьные и строковые величины

Символьные величины

Литерный (символьный) тип char определяется множеством значений кодовой таблицы ПК. Каждому символу задается целое число от 0 до 255. В программе значения переменных и констант типа char должны быть заключены в апострофы.

Над данными символьного типа определены операции отношения: =, <>, >,<,<=,>=, вырабатывающие результат логического типа, и следующие стандартные функции:

Chr(x) – преобразует выражение x в символ и возвращает значение символа

Ord(ch) – преобразует символ ch в его код и возвращает значение кода

Pred(ch) – возвращает предыдущий символ

Succ(ch) – возвращает следующий символ

Пример.

Ord(‘:’) =58

Ord(‘A’)=65

Chr(128)=Б

Pred(‘Б’)=A

Succ(‘Г’)=Д

Строковые величины

Строка (строковый тип данных) – это последовательность символов кодовой таблицы ПК. Количество символов в строке (длина строки) может

лежать в диапазоне от 0 до 255. Для определения данных строкового типа используется идентификатор string, за которым следует значение максимальной длины строки данного типа (заключается в квадратные скобки).

Строковые данные могут использоваться в качестве констант. Строковая константа – последовательность символов, заключенная в апострофы. Например, ‘237’, ‘это строковая константа’.

Переменную строкового типа можно определить в разделе описания переменных:

Var <имя>: string[<максимальная длина строки>].

Например, var Name: string[20]. В описании строки можно не указывать длину, в этом случае она равна максимальной величине – 255. Элементы строки определяются именем строки с индексом, заключенным в квадратные скобки. Например, N[5]. Первый символ строки имеет номер 1 и т.д. Можно сказать, что строка представляет собой одномерный массив, элементами которого являются символы. Тип string и тип char совместимы, они могут употребляться в одних и тех же выражениях.

Выражения, в которых операндами служат строковые данные, называются строковыми. Они могут состоять из строковых констант, переменных, знаков операций. Над этими данными допустимы операция сцепления (конкатенация) и операции отношения.

Операция сцепления (+) применяется для соединения нескольких строк в одну строку. Сцеплять можно и константы, и переменные. Длина результирующей строки не должна превышать 255 символов.

Операции отношения (=, <>, >, <, <=, >=) проводят сравнение двух строк и имеют приоритет более низкий, чем операция конкатенации. Сравнение строк производится слева направо до первого несовпадающего символа. Строка считается больше, если в ней первый несовпадающий символ имеет больший номер в таблице кодов.

Например, ‘MS-DOS’<’MS-Dos’.

Если строки имеют различную длину, но в общей части символы совпадают, то более короткая строка меньше.

Например, ‘Компьютер’<’Компьютер ’.

Строки равны, если они полностью совпадают.

Например, ‘Маска’ = ’Маска’.

Для обработки строковых данных можно использовать специальные процедуры и функции.

Процедура Delete(St, poz, n) – удаление n символов строки St, начиная с позиции Poz.

Пример

Значение St	Выражение	Результат
‘абвгде’	Delete(St, 4, 2);	‘абве’
‘река Волга’	Delete(St, 1, 5);	‘Волга’

Процедура Insert (S_1, S_2, Poz) – вставка строки S_1 в строку S_2, начиная с позиции Poz.

Пример

Значение S_1	Значение S_2	Оператор	Результат
‘ ЕС ’	‘ЭВМ1841’	Insert (S_1, S_2, 4);	‘ЭВМ ЕС 1841’
‘ N’	‘ Рис. 2’	Insert (S_1, S_2 , 6);	‘ Рис. N2’

Процедура Str(N,St) – преобразование числового значения N в строковый и помещение результата в строку St.

Пример

Значение N	Выражение	Результат
1500	Str (N:6,St);	‘ 1500’

Процедура Val(St, N,Code) – преобразует значение St в величину целочисленного или вещественного типа и помещает результат в N. Code – целочисленная переменная. Если во время операции преобразования ошибки не обнаружено, значение Code равно 0, если же обнаружена ошибка, то Code будет содержать номер позиции первого ошибочного символа, а значение N не определено.

Пример

Значение St	Выражение	Результат
‘1500’	Val(St, N,Code);	Code=0
’14.2A+02	Val(St, N,Code);	Code=5

Функция Copy(S, Poz, N) – выделяет из строки S подстроку длиной N символов, начиная с позиции Poz.

Пример

Значение S	Выражение	Результат
‘Мама мыла раму’	Copy(S, 6, 4);	‘мыла’

Функция Concat(S_1,S2,…,S_n) – выполняет сцепление строк S_1,S2,…,S_n в одну строку.

Пример

Выражение	Результат
Concat(‘Мама ’,’мыла ’,’раму’)	‘Мама мыла раму’

Функция Length(S) – определяет текущую длину строки S.

Пример

Значение S	Выражение	Результат
‘1500 символов’	Length(S)	13

Функция Pos(S_1,S_2) – определяет первое появление в строке S_2 подстроки S_1.

Пример

Значение S_2	Выражение	Результат

‘abcdef’	Pos(‘cd’,S_2)	3
‘abcdef’	Pos(‘k’,S_2)	0

Функция UpCase (ch) – преобразует строчную букву в прописную.Обрабатывает буквы только латинского алфавита.
Пример

Значение Ch	Выражение	Результат
‘d’	UpCase (ch)	‘D’

5.3 Записи и множества

Записи (тип-запись)
Запись обозначается в Паскале record. Этот тип служит введением в объектно-ориентированное программирование, как следствие приблизит к программированию под Windows.
Работа с записями
Рассмотрим конкретную задачу. Пусть нам необходимо создать записную книжку с адресами, которая будет некоторым подобием базы данных. Суть программы будет в следующем:

- Программа может спрашивать и выдавать следующие данные:

1. Фамилия, имя, отчество;
2. Адрес: улица, дом;
3. Телефон;
4. E-mail;

- Пусть количество адресатов будет ограничено 10-ю адресами.

Массив может хранить только одну переменную в каждом своем элементе. В принципе, нам и требуется что-нибудь вроде массива, только необходимо, чтобы в каждом его элементе сохранялось 4 переменных. Понятно, что с помощью массивов такого сделать нельзя.
Здесь мы и подходим к понятию записей. На самом деле запись - это контейнер, в котором именно "записано" несколько переменных. Чем-то похоже на массив, только переменные могут быть разных типов, каждая из них имеет свое имя.
Для начала определим типы для наших данных (адрес):

- Fio: String; {фио}
- Adress: String: {адрес}
- Phone: LongInt; {телефон, в виде числа}
- Email: String; {e-mail}

Переменных несколько, все они имеют имена и разных типов.
Запись - это способ объединения нескольких переменных разных типов в одной. Благодаря этому достигается замечательная упорядоченность данных, программы при этом упрощаются и становятся логичнее.

Записи описываются в разделе type, который подобен разделам var или const. В этом разделе описываются все типы, определяемые пользователем. Описываются записи с помощью служебного слова record, перед которым идет имя записи:

```
type
AdressItem = Record
```

После описываются все переменные, которые будут содержаться в записи, подобно тому, как они описываются в разделе var. Завершается запись словом end;. Вот пример записи, необходимой для нашей программы:

```
type
AdressItem = Record
Fio: String;
Adress: String;
Phone: LongInt;
Email: String;
end;
```

Создав в программе запись, вы получаете новый тип и нужно создать переменные этого типа. Здесь тоже нет ничего сложного. После того, как вы опишете запись в разделе type, можно создать переменные нового типа в разделе var:

```
var
A,B: AdressItem;
```

По условию нам нужно 10 записей (у нас адресная книжка на 10 адресов). Сделаем массив из созданной записи:

```
var
Book: Array [1..10] of AdressItem;
```

При этом каждый элемент массива Book в свою очередь содержит другие, собственные переменные. Существует два способа обращения к элементам записи:

1. Поля записи (ее внутренние переменные) могут быть изменены путем использования служебного слова with. При этом строится небольшая конструкция, внутри которой и происходят все манипуляции с записью. Например, (читаем значения в запись A):

```
var
A: AdressItem;
begin
with A do
Write('ФИО: ');
Readln(Fio);
Write('Адрес: ');
Readln(Adress);
Write('Телефон: ');
```

```
Readln(Phone);
end;
end.
```

Синтаксис конструкции with:

```
    with _имя_записи_ do
    .... действия с ее полями (ее переменными) .....
    end;
```

2. Второй способ обращения к полям записи. Суть его в следующем:

- К полю записи можно обратиться, указав имя записи и через точку имя поля.

Например,

```
var
A: AdressItem;
begin
Write('ФИО: ');
Readln( A.Fio );
Write('Адрес: ');
Readln( A.Adress );
Write('Телефон: ');
Readln( A.Phone );
end.
```

Тип -множество

Множества - это наборы однотипных логически связанных друг с другом объектов. Характер связей между объектами лишь подразумевается программистом и никак не контролируется Турбо Паскалем. Количество элементов, входящих во множество, может меняться в пределах от 0 до 256 (множество, не содержащее элементов, называется пустым). Именно непостоянством количества своих элементов множества отличаются от массивов и записей.

Два множества считаются эквивалентными тогда и только тогда, когда все их элементы одинаковы, причем порядок следования элементов в множестве безразличен. Если все элементы одного множества входят также и в другое, говорят о включении первого множества во второе. Пустое множество включается в любое другое.

Пример определения и задания множеств:

```
type
digitChar= set of '0'..'9';
digit = set of 0. .9;
var
sl,s2,s3 :digitChar;
s4,s5,s6 :digit;
```

```
begin
.....
s1:=['1','2','3'];
s2:=['3','2','1'];
s3:=['2','3'];
s4:=[0..3,6];
s5:=[4,5];
s6:=[3..9];
.....
end.
```

В этом примере множества S1 и S2 эквивалентны, а множество S3 включено в S2 , но не эквивалентно ему.

Описание типа множества имеет вид:

<имя типа> = SET OF <баз.тип>

Здесь <имя типа> - правильный идентификатор;

SET, OF - зарезервированные слова (множество, из);

<баз.тип> - базовый тип элементов множества, в качестве которого может использоваться любой порядковый тип, кроме WORD, INTEGER, LONGINT.

Для задания множества используется так называемый конструктор множества: список спецификаций элементов множества, отделяемых друг от друга запятыми; список обрамляется квадратными скобками. Спецификациями элементов могут быть константы или выражения базового типа, а также - тип-диапазон того же базового типа.

Над множествами определены следующие операции:

* пересечение множеств; результат содержит элементы, общие для обоих множеств; например, S4*S6 содержит [3], S4*S5 - пустое множество (см. выше);

\+ объединение множеств; результат содержит элементы первого множества, дополненные недостающими элементами из второго множества:

S4+S5 содержит [0,1,2,3,4,5,6];

S5+S6 содержит [3,4,5,6,7,8,9];

\- разность множеств; результат содержит элементы из первого множества, которые не принадлежат второму:

S6-S5 содержит [3,6,7,8,9];

S4-S5 содержит [0,1,2,3,6];

= проверка эквивалентности; возвращает TRUE, если оба множества эквивалентны;

<> проверка неэквивалентности; возвращает TRUE, если оба множества неэквивалентны;

<= проверка вхождения; возвращает TRUE, если первое множество включено во второе;

>= проверка вхождения; возвращает TRUE, если второе множество включено в первое;
IN проверка принадлежности; в этой бинарной операции первый элемент - выражение, а второй - множество одного и того же типа; возвращает TRUE , если выражение имеет значение, принадлежащее множеству:
3 in s6 возвращает TRUE;
2*2 in s1 возвращает FALSE.
Дополнительно к этим операциям можно использовать две процедуры.
INCLUDE - включает новый элемент во множество. Обращение к процедуре:
INCLUDE (S,I)
Здесь S - множество, состоящее из элементов базового типа TSetBase;
I - элемент типа TSetBase, который необходимо включить во множество.
EXCLUDE - исключает элемент из множества. Обращение:
EXCLUDE(S,I)
Параметры обращения - такие же, как у процедуры INCLUDE.
В отличие от операций + и -, реализующих аналогичные действия над двумя множествами, процедуры оптимизированы для работы с одиночными элементами множества и поэтому отличаются высокой скоростью выполнения.

5.4 Файловые типы

Работа с файлами
В Паскале работа с файлами осуществляется через специальные типы. Это файловые типы, которые определяют тип файла, то есть фактически указывают его содержимое. С помощью этой переменной, которой присвоен необходимый тип, и осуществляется вся работа с файлами - открытие, запись, чтение, закрытие и т.п.
При работе с файлами существует определенный порядок действий, которого необходимо придерживаться. Вот все эти действия:
1. Создание (описание) файловой переменной;
2. Связывание этой переменной с конкретным файлом на диске или с устройством ввода-вывода (экран, клавиатура, принтер и т.п.);
3. Открытие файла для записи либо чтения;
4. Действия с файлом: чтение либо запись;
5. Закрытие файла.
Возможно связать файловую переменную не только с физическим файлом на носителе информации, но и с устройством. В качестве такового используются обычные псевдонимы устройств DOS. Вот основные два:

1. CON - консоль (экран-клавиатура), то есть по записи в это устройство мы будем получать информацию на экран, при чтении информации из этого устройства, будем читать данные с клавиатуры.
2. PRN - принтер. При записи в это устройство вы получите информацию на принтер.

Последний этап - закрытие файла. В принципе, не обязательное условие для файлов, из которых мы читаем данные. Если не закроем - ошибки это не вызовет, последствий тоже. Однако обязательно закрывать файл, если мы осуществляли в него запись. Дело в том, что если мы пишем данные в файл на диске и забываем его закрыть, - информация не сохранится. Она (информация) помещается во временный буфер, который запишется на диск только при закрытии файла.

Типы файловых переменных

В Turbo Pascal имеется три типа таких переменных, которые определяют тип файла. Вот эти типы:

1. Text - текстовый файл. Из переменной такого типа мы сможем читать строки и символы.
2. File of _любой_тип_ - так называемые "типизированные" файлы, то есть файлы, имеющие тип. Этот тип определяет, какого рода информация содержится в файле и задается в переметре _любой_тип_. К примеру, если мы напишем так:

F: File of Integer;

То Паскаль будет считать, что файл F содержит числа типа Integer; Соответственно, читать из такого файла мы сможем только переменные типа Integer, равно как и писать.

3. File - нетипизированный файл. Когда мы указываем в качестве типа файла просто File, то есть без типа:

F: File;

То получаем "нетипизированный" файл, чтение и запись в который отличается от работы с файлами других типов. Эти действия производятся путем указания количества байт, которые нужно прочитать, а также указанием области памяти, в которую нужно прочитать эти данные.

Связывание переменной с файлом

Выполняется одной и той же процедурой для всех типов файлов - Assign:

Assign(_переменная_файлового_типа_, 'путь к файлу');

В качестве параметров задаются переменная любого файлового типа и строка - путь к файлу, который, по правилам DOS, не может быть длиннее 79 символов. Вот пример использования assign:

```
var
  T: Text;
  F1: File of Byte;
  F2: File;
```

```
begin
  Assign(T, '1.txt');
  Assign(F1, 'C:\Programm\Tp\2.txt');
  Assign(F2, 'F:\usr\bin\apache\conf\httpd~1.con');
  ..........
```

Открытие файла

Открытие файла - это уже более усложненный процесс, нежели связывание с ним переменной. Здесь учитывается, зачем открывается файл - для записи или чтения, а также в зависимости от типа файла процедуры выполняют различные действия.

Этот процесс заключается в использовании одной из трех имеющихся процедур:

1. Reset(_любая_файловая_переменная_);

Открывает файл на чтение. В качестве параметра - файловая переменная любого из перечисленных выше типов. Это может быть текстовый, типизированный либо не типизированный файл. В случае с текстовым файлом, он открывается только на чтение. В случае с типизированным и нетипизированным файлом - он открывается на чтение и запись.

2. Append(T: Text);

Эта процедура открывает текстовый файл (только текстовый!) на запись. То есть если вы используете текстовый файл и хотите производить в него запись, нужно использовать Append. Если чтение - Reset. В остальных случаях дело обходиться одной процедурой Reset.

Также обратите внимание, что если вы до этого уже открыли файл на чтение, вам не нужно закрывать его и открывать снова на запись. В этом случае файл закрывается сам и открывается заново. При записи данных в файл при открытии его с помощью этой процедуры они записываются в конец файла.

3. ReWrite(F) - создает новый файл либо перезаписывает существующий. Файл, открытый с помощью этой процедуры будет полностью перезаписан.

Способ проверки наличия файла на диске заключается в двух этапах: использовании ключей компилятора и функции IOResult, которая возвращает значение от только что выполненной операции ввода-вывода.

Ключи компилятора - это переключатели, которые контролируют ход выполнения программы исключая или включая реакцию на какие-нибудь условия. В нашем случае нас интересует условие, когда физически отсутствует нужный нам файл, либо не удалось открыть его по другим причинам.

Оформляются ключи следующим образом: в скобках комментариев "{}" первым символом после открывающей скобки "{" ставиться знак доллара "$", после чего указывается имя ключа и его значение. Например:

{$I+} - включение вывода ошибок

{$I-} - выключение вывода ошибок

Т.е. отсутствие файла - это ошибка, которая возвращается функцией IOResult. Если же эта функция возвращает 0, то файл успешно открыт, без ошибок. Последовательность действий, необходимых для проверки на наличие файла:

1. Связываем переменную с файлом;
2. Выключаем вывод ошибок на экран - {$I-}
3. Открываем файл необходимой нам процедурой;
4. Включаем вывод ошибок {$I+} - для дальнейшего отслеживания таковых;
5. Проверяем, если IOResult возвращает нуль, то файл открыт. Иначе выводим ошибку.

Вот пример такой программы:

```
var
  T: Text;
  S: String;
begin
  Write('Enter filename: ');
  Readln(S);
  Assign(T, S);
{$I-}
  Reset(T);   { открываем файл для чтения }
{$I+}
  if IOResult <> 0 then   { если не нуль, то была ошибка }
  begin
    Write('Error when open file!');
    Halt;
  end;
  { иначе все в порядке, продолжаем }
  ..........
end.
```

Закрытие файла

Закрытие файла производится с помощью процедуры Close(F), где F - это переменная файлового типа. Эта процедура одна для всех типов файлов.

Запись и чтение файлов.

Чтение файлов. Чтение файлов производится с помощью процедур Read и Readln. Перед переменной, в которую помещается считанное значение, указывается переменная файлового типа (дескриптор файла):

Read(F, C);

Здесь F - дескриптор файла, C - переменная (Char, String - для текстовых, любого типа - для типизированных файлов).

Самой главной функции при чтении файлов является функция поверки на конец файла - Eof(F): Boolean;. В качестве параметра - файловая переменная любого типа. Функция возвращает TRUE если достигнут конец файла и FALSE иначе.

Запись в файлы. Запись в файлы производится с помощью процедур Write и Writeln. Как и в случае с чтением, перед записываемой в файл переменной указывается дескриптор файла:

Write(F, S);

Здесь F - дескриптор, S - переменная.

При этом, естественно, переменная должна соответствовать типу файла.

"Нетипизированные" файлы, то есть файлы без типа.

Работа с этими файлами несколько отличается от работы с текстовыми и типизированными. При действиях с файлами без типа, мы не знаем, что за данные в них находятся и в переменные какого типа их надо помещать.

Суть нетипизированных файлов в следующем: имея файл без определенного типа, мы можем читать из него любые данные, будь то строки, символы или записи. Читая данные из файла без типа, мы получаем блоки информации, которые составляют обычный набор байт. Указывая переменную, в которую эти байты надо поместить, мы как бы "на ходу преобразуем" эти данные к нужному типу.

Чтение из файлов без типа

Процедура связывания файловой переменной с внешним файлом и его открытие ничем чем отличаются от обычного порядка действий. Разве что переменная в данном случае должна иметь тип File; , то есть быть файлом без типа.

```
var
  F: File;
begin
  { связываем файл с переменной }
  Assign(F, '1.txt');
  Reset(F);
end.
```

Чтение производится с помощью процедуры BlockRead.

BlockRead(F: File, Buf: Var, Size: Word, Result: Word)

F: File; - переменная типа File; Именно из этой переменной и происходит чтение данных.

Buf: Var; - переменная любого типа. В эту переменную помещаются прочитанные данные.

Size: Word; - количество считываемых байт.

Result: Word; - в эту переменную помещается реальное количество байт, которые были прочитаны.

Работает эта процедура следующим образом: из файла F считывается Size записей, которые помещаются в память, начиная с первого байта переменной Buf. После выполнения процедуры реальное количество прочитанных байт помещается в переменную Result. Эта переменная совсем не обязательно должна присутствовать в качестве параметра, то есть ее попросту можно опустить. Однако иногда она необходима - например, если чтение было окончено до того, как было прочитано требуемое количество байт (достигнут конец файла), мы можем это отследить через переменную Result. Если же в этом случае (чтение данных после конца файла) переменная Result не будет указана, то образуется ошибка времени выполнения N100 "Disk read error" (Runtime error 100).
Функция Sizeof - принимает в качестве параметра любую переменную и возвращает ее размер в байтах.
Дополнительный параметр процедуры Reset. Он указывает размер буфера, который используется для передачи данных. С текстовыми файлами он не используется.
Буфер по умолчанию равен 128 байт. Если его явно не указывать, то Паскаль устанавливает это значение.
Запись в файлы без типа
Для этого в Паскале имеется еще одна, отдельная процедура, а именно BlockWrite. BlockWrite(F: File, Buf: Var, Size: Word, Result: Word)
F: File; - переменная типа File;
Buf: Var; - переменная любого типа. Начиная с этой переменной, данные будут записываться в файл.
Size: Word; - количество записываемого блока данных в байтах.
Result: Word; - в эту переменную помещается реальное количество байт, которые были записаны.
Функция ParamStr. Эта функция возвращает параметр командной строки под номером, который ей задается. К примеру, если данная программа запускается так:
C:\copy.exe 1.txt A:\1.txt
То функция ParamStr(1) вернет строку "1.txt". Функция ParamStr(2) - строку "A:\1.txt".

5.5 Совместимость и преобразование типов

Турбо Паскаль - это типизированный язык. Он построен на основе строгого соблюдения концепции типов, в соответствии с которой все применяемые в языке операции определены только над операндами совместимых типов. Ниже приводится более полное определение совместимости типов.
Два типа считаются совместимыми, если:
оба они есть один и тот же тип;

оба вещественные;
оба целые;
один тип есть тип-диапазон второго типа;
оба являются типами-диапазонами одного и того же базового типа;
оба являются множествами, составленными из элементов одного и того же базового типа;
оба являются упакованными строками (определены с предшествующим словом PACKED) одинаковой максимальной длины;
один тип есть тип-строка, а другой - тип-строка, упакованная строка или символ;
один тип есть любой указатель, а другой - нетипизированный указатель;
один тип есть указатель на объект, а другой - указатель на родственный ему объект;
оба есть процедурные типы с одинаковыми типом результата (для типа-функции), количеством параметров и типом взаимно соответствующих параметров.
Совместимость типов приобретает особое значение в операторах присваивания. Пусть T1 - тип переменной, а T2 - тип выражения, т.е. выполняется присваивание T1 := T2. Это присваивание возможно в следующих случаях:
T1 и T2 есть один и тот же тип и этот тип не относится к файлам или массивам файлов, или записям, содержащим поля-файлы, или массивам таких записей;
T1 и T2 являются совместимыми порядковыми типами и значение T2 лежит в диапазоне возможных значений T1;
T1 и T2 являются вещественными типами и значение T2 лежит в диапазоне возможных значений T1;
T1 - вещественный тип и T2 - целый тип; ,
T1 - строка и T2 - символ;
T1 - строка и T2 - упакованная строка;
T1 и T2 - совместимые упакованные строки;
T1 и T2 - совместимые множества и все члены T2 принадлежат множеству возможных значений T1;
T1 и T2 - совместимые указатели;
T1 и T2 - совместимые процедурные типы;
T1 - объект и T2 - его потомок.
В программе данные одного типа могут преобразовываться в данные другого типа. Такое преобразование может быть явным или неявным.
При явном преобразовании типов используются вызовы специальных функций преобразования, аргументы которых принадлежат одному типу, а значение - другому. Таковыми являются уже рассмотренные функции ORD, TRUNC, ROUND, CHR. В гл. 6 описывается функция PTR, преобразующая четырехбайтный целочисленный аргумент к типу-указателю.

В Турбо Паскале может использоваться и более общий механизм преобразования типов, согласно которому преобразование достигается применением идентификатора имени) стандартного типа или типа, определенного пользователем, как идентификатора функции преобразования к выражению преобразуемого типа (так называемое автоопределенное преобразование типов). Например, допустимы следующие вызовы функций:

```
type
MyType = (a, Ь, c, d);
.....
MyType (2)
Integer ('D')
pointer (longint(a)+ $FF)
Char (127 mod c)
Byte (k)
```

При автоопределенном преобразовании типа выражения может произойти изменение длины его внутреннего представления (длина может увеличиться или уменьшиться).

В Турбо Паскале определен еще один явный способ преобразования данных: в ту область памяти, которую занимает переменная некоторого типа, можно поместить значение выражения другого типа, если только длина внутреннего представления вновь размещаемого значения в точности равна длине внутреннего представления переменной. С этой целью вновь используется автоопределенная функция преобразования типов, но уже в левой части оператора присваивания:

```
type
byt = array [1..2] of Byte;
int = array [1..2] of Integer;
rec = record
x, y : Integer
end;
var
vbyt : byt;
vint : int;
vrec : rec;
begin
byt(vint[1])[2] := 0;
int(vrec)[1] := 256
end.
```

Неявное преобразование типов возможно только в двух случаях:

в выражениях, составленных из вещественных и целочисленных переменных, последние автоматически преобразуются к вещественному типу, и все выражение в целом приобретает вещественный тип;

одна и та же область памяти попеременно трактуется как содержащая данные то одного, то другого типа (совмещение в памяти данных разного типа).

Совмещение данных в памяти может произойти при использовании записей с вариантными полями, типизированных указателей, содержащих одинаковый адрес, а также при явном размещении данных разного типа по одному и тому же абсолютному адресу. Для размещения переменной по нужному абсолютному адресу она описывается с последующей стандартной директивой ABSOLUTE, за которой помещается либо абсолютный адрес, либо идентификатор ранее определенной переменной. Абсолютный адрес указывается парой чисел типа WORD, разделенных двоеточием; первое число трактуется как сегмент, второе - как смещение адреса. Например:

```
b : Byte absolute $0000:$0055; w : Longlnt absolute 128:0;
```

Если за словом ABSOLUTE указан идентификатор ранее определенной переменной, то происходит совмещение в памяти данных разного типа, причем первые байты внутреннего представления этих данных будут располагаться по одному и тому же абсолютному адресу, например:

```
var
x : Real;
y : array [1..3] of Integer absolute x;
```

В этом примере переменные X и Y будут размещены, начиная с одного и того же абсолютного адреса. Таким образом, одну и ту же область памяти длиной 6 байт, а, следовательно, и размещенные в этой области данные теперь можно рассматривать как данные либо типа REAL, либо как массив из трех данных типа INTEGER. Например, следующая программа выдаст на экран содержимое первых двух байт внутреннего представления вещественного числа пи = 3.1415 в виде целого числа:

```
var
x : Real; y : array[1..3] of Integer absolute x;
begin
x := pi; WriteLn(y[1])
end.
```

На экран будет выдан результат 8578.

Неявные преобразования типов могут служить источником трудно обнаруживаемых ошибок в программе, поэтому везде, где это возможно, следует избегать их.

5.6 Динамическая память

Все переменные, объявленные в программе, размещаются в одной непрерывной области оперативной памяти, которая называется сегментом дан-

ных. Длина сегмента данных определяется архитектурой микропроцессоров 80x86 и составляет 65536 байт, что может вызвать известные затруднения при обработке больших массивов данных. С другой стороны, объем памяти ПК (обычно не менее 640 Кбайт) достаточен для успешного решения задач с большой размерностью данных. Выходом из положения может служить использование так называемой динамической памяти.
Динамическая память - это оперативная память ПК, предоставляемая программе при ее работе, за вычетом сегмента данных (64 Кбайт), стека (обычно 16 Кбайт) и собственно тела программы. Размер динамической памяти можно варьировать в широких пределах. По умолчанию этот размер определяется всей доступней памятью ПК и, как правило, составляет не менее 200...300 Кбайт.
Динамическая память - это фактически единственная возможность обработки массивов данных большой размерности. Многие практические задачи трудно или невозможно решить без использования динамической памяти. Такая необходимость возникает, например, при разработке систем автоматизированного проектирования (САПР): размерность математических моделей, используемых в САПР, может значительно отличаться в разных проектах; статическое (т.е. на этапе разработки САПР) распределение памяти в этом случае, как правило, невозможно. Наконец, динамическая память широко используется для временного запоминания данных при работе с графическими и звуковыми средствами ПК.
Динамическое размещение данных означает использование динамической памяти непосредственно при работе программы. В отличие от этого статическое размещение осуществляется компилятором Турбо Паскаля в процессе компиляции программы. При динамическом размещении заранее не известны ни тип, ни количество размещаемых данных, к ним нельзя обращаться по именам, как к статическим переменным.

Вопросы для самоподготовки:

1. В чем различие и сходство данных типа char и string?
2. В чем различие и сходство данных типа массив и множество?
3. В чем различие и сходство данных типа массив и запись?
4. Какую роль в программе играет файловая переменная?
5. Для чего вводят динамические структуры?

Глава 6. Процедуры и функции

6.1 Ввод и вывод данных

Решение даже самой простой задачи на компьютере не обходится без операций ввода – вывода информации. Ввод данных – это передача информа-

ции от внешнего носителя в оперативную память для обработки. Вывод - обратный процесс, когда данные передаются после обработки из оперативной памяти на внешний носитель (экран монитора, принтер, дискету или винчестер и другие устройства). Выполнение этих операций производится путем обращения к стандартным процедурам: Read, Readln, Write, Writeln.

Ввод данных с клавиатуры

Процедура чтения Read обеспечивает ввод данных для последующей их обработки программой. Общий вид: Read (<список переменных>);
В списке перечисляются имена переменных. Значения этих переменных набираются через пробел на клавиатуре и высвечиваются на экране после запуска программы. После набора данных для одной процедуры Read нажимается клавиша ввода Enter. Значения переменных должны вводиться в строгом соответствии с синтаксисом языка Паскаль. Если соответствие нарушено, то возникают ошибки.

Процедура чтения Readln аналогична процедуре Read, единственное отличие в том, что после считывания последнего в списке значения курсор переходит на начало новой строки.

Пример:

```
Program primer;
Var   i, k:integer;  c,d, s: real;
begin
readln (c,d);
read(i,k);
…
end.
```

В данном случае необходимо ввести сначала два действительных числа через пробел. Переменной c присваивается значение, равное первому введенному числу, а переменной d – значение, равное второму введенному числу. После ввода этих значений курсор переходит на начало новой строки (за это отвечает ln следующий за словом Read). Далее требуется ввести еще два целых числа, которые будут присвоены значениям переменных i и k соответственно.

Вывод данных

Процедура вывода Write производит вывод данных.
Общий вид: Write(<список вывода>);

В списке вывода могут быть представлены выражения допустимых типов данных (integer, real, char и т.д.) и произвольный текст, заключенный в апострофы.

Например, Write('Привет'); Write(34.7); Write(45+55); Write(b, d);

Процедура Writeln аналогична процедуре Write. Отличие в том, что после вывода последнего в списке выражения курсор переходит на начало новой строки.

В процедурах вывода Write и Writeln имеется возможность записи выражения, определяющего ширину поля вывода.

При рассмотрении форматов вывода примем следующие обозначения:
I, p, q – целочисленное выражение;
R - выражение вещественного типа;
- цифра;
* - знак «+» или «- »;
_ - пробел.

Значение I	Выражение	Результат
324	Write (I);	324
34	Write (I, I, I);	343434
324	Write (I : 6);	___324
312	Write (I + I : 7);	____624
Значение R	Выражение	Результат
123.432	Write (R);	__1.2343200000E+02
-1.34E+01	Write (R);	_ -1.3400000000E+01
304.55	Write (R :15);	3.045500000E+02
Значение R	Выражение	Результат
304.66	Write (R :8 : 4);	304.6600
45.322	Write (R : 5 : 2);	45.32

6.2 Процедуры

Для использования подпрограммы-процедуры необходимо сначала описать процедуру, а затем обращаться к ней (обращение к процедуре – отдельный оператор). Описание процедуры включает заголовок (имя) и тело процедуры. Заголовок состоит из зарезервированного слова procedure, имени процедуры и, заключенного в скобки, списка формальных параметров с указанием типа. Название «формальные» эти параметры получили в связи с тем, что в этом списке заданы только имена для обозначения исходных данных и результатов работы процедуры, а при вызове подпрограммы на их место будут поставлены конкретные значения. Тело процедуры – блок, по структуре аналогичный программе.

При создании программ, использующих процедуры, следует учитывать, что все объекты, которые описываются после заголовка в теле проце-

дуры, называются локальными объектами и доступны только в пределах этой процедуры.

Все объекты, описанные в вызывающей программе, называются глобальными и являются доступными внутри процедур, вызываемых этой программой.

Общий вид описания процедуры:

```
Procedure <имя> (список формальных параметров, блок описания);
Const …;  ⎤
   …      ⎬ блок описания
Var ….;   ⎦
begin
<операторы>
end;
```

6.3 Функции

Подпрограмма-функция обрабатывает данные, переданные ей из главной программы, и затем возвращает полученный результат (в отличие от процедуры). Функция, определенная пользователем, состоит из заголовка и тела функции. Заголовок содержит зарезервированное слово Function, имя, список формальных параметров (заключенный в скобки) и тип возвращаемого функцией значения. Тело функции представляет собой локальный блок, по структуре сходный с программой. Общий вид описания функции:

```
   Function <имя> (<параметры>): <тип результата>;
Const …;  ⎤
   …      ⎬ блок описания
Var ….;   ⎦
begin
<операторы>
end;
```

В разделе операторов должен находиться, хотя бы один оператор, присваивающий имени функции значение. Обращение к функции осуществляется по имени с указанием списка аргументов. Каждый аргумент должен соответствовать формальным параметрам и иметь тот же тип.

6.4 Рекурсия

Рекурсия - это такой способ организации вычислительного процесса, при котором подпрограмма в ходе выполнения составляющих ее операторов обращается сама к себе.

Классический пример - вычисление факториала. Программа вводит с клавиатуры целое число N и выводит на экран значение N!, которое вы-

числяется с помощью рекурсивной функции. При выполнении правильно организованной рекурсивной подпрограммы осуществляется многократный переход от некоторого текущего уровня организации алгоритма к нижнему уровню последовательно до тех пор, пока, наконец, не будет получено тривиальное решение поставленной задачи.

Рекурсивная форма организации алгоритма обычно выглядит изящнее итерационной и дает более компактный текст программы, но при выполнении, как правило, медленнее и может вызвать переполнение стека (при каждом входе в подпрограмму ее локальные переменные размещаются в особым образом организованной области памяти, называемой программным стеком). Переполнение стека особенно ощутимо сказывается при работе с сопроцессором: если программа использует арифметический сопроцессор, результат любой вещественной функции возвращается через аппаратный стек сопроцессора, рассчитанный всего на 8 уровней. Чтобы избежать переполнения стека сопроцессора, следует размещать промежуточные результаты во вспомогательной переменной.

Вопросы для самоподготовки.

1. В чем принципиальные отличия подпрограммы-процедуры от подпрограммы-функции?
2. Что такое рекурсия?
3. Как осуществляется обращение к функции в программе?

Глава 7. Стандартные библиотечные модули

В систему Турбо Паскаль включены 8 модулей: System, Crt, Dos, Graph, Graph3, Overlay, Printer, Turbo3 и специализированная библиотека Turbo Vision. Модуль System подключается по умолчанию, поэтому в любой программе становятся доступными все его встроенные процедуры и функции. Остальные модули должны подключаться с помощью зарезервированного слова uses с добавлением имени модуля. Например: uses Crt.

Рассмотрим кратко назначение каждого модуля.

System - сердце Турбо Паскаля. Подпрограммы, содержащиеся в нем, обеспечивают работу всех остальных модулей системы.

Crt - содержит средства управления дисплеем и клавиатурой компьютера.

Dos - включает средства, позволяющие реализовывать различные функции Dos.

Graph3 - поддерживает использование стандартных графических подпрограмм.

Overlay - содержит средства организации специальных оверлейных программ.

Printer - обеспечивает быстрый доступ к принтеру.

Turbo3 - обеспечивает максимальную совместимость с версией Турбо Паскаль 3.0.
Graph - содержит пакет графических средств.
Turbo Vision - библиотека объектно-ориентированных программ для разработки пользовательских интерфейсов.

7.1 Модуль Crt

CRT - стандартный модуль Turbo Pascal, в котором содержатся разнообразные средства ввода-вывода. Процедуры и функции CRT помогут организовать хороший интерфейс в программах. Кроме процедур и функций любой модуль может также содержать описание типов, константы и переменные, доступные в пользовательской программе (если имя модуля указано в операторе USES). В модуле CRT определена переменная

VAR TextAttr : BYTE ,

в ней содержится текущий цвет фона и цвет символов, используемые при выводе на экран процедурами WRITE и WRITELN. Изменив эту переменную, вы задаете новый цветовой атрибут. Цветовой атрибут строится следующим образом: в четырех младших битах хранится цвет символов (от 0 до 15), в следующих трех битах - цвет фона (от 0 до 7), и старший бит отвечает за мерцание. Пусть, например, значение переменной TextAttr равно 237, в двоичной записи - это 1110 1101 (если записывать биты от старшего к младшему). Четыре младших бита (1101) дают цвет символов 13, или LightMagenta - светло малиновый; следующие 3 бита (110) дают цвет фона 6, или Brown - коричневый, старший бит - единичный. Таким образом, будут выводиться мерцающие светло-малиновые символы на коричневом фоне. Теперь рассмотрим некоторые функции и процедуры модуля CRT:
1. FUNCTION KeyPressed : Boolean - возвращает TRUE, если буфер клавиатуры не пуст (все нажатия клавиш во время работы программы накапливаются в специальном участке памяти - буфере клавиатуры, откуда затем поступают в программу). Функция не очищает буфер клавиатуры.
2. FUNCTION ReadKey : Char - считывает символ из буфера клавиатуры, если буфер пуст, то ожидает нажатия клавиши. Эту функцию удобно использовать для организации пауз в программе.
3. PROCEDURE Delay(MS: Word) - приостанавливает выполнение программы на MS миллисекунд.
4. PROCEDURE Sound(Hz: Word) - генерирует звуковой сигнал с частотой Hz герц.
5. PROCEDURE NoSound - выключает звуковой сигнал.
6. PROCEDURE Window(X1,Y1,X2,Y2:Byte) - определяет на экране текстовое окно, заданное координатами верхнего левого и нижнего правого

углов. Текстовое окно - это прямоугольная область на экране, куда направляется весь вывод. Процедура не выполняет никаких видимых действий.
7. PROCEDURE TextBackground(Color: Byte) - задает цвет фона для всего последующего вывода.
8. PROCEDURE TextColor(Color: Byte) - задает цвет символов для всего последующего вывода. Процедуры TextBackground и TextColor вместе обеспечивают те же возможности, что и переменная TextAttr.
9. PROCEDURE ClrScr - очищает текущее окно, используя текущий фоновый цвет.
10. PROCEDURE GotoXY(X,Y:Byte) - перемещает курсор в позицию X строки Y текущего окна. Координаты отсчитываются от левого верхнего угла окна.
11. FUNCTION WhereX : Byte и
12. FUNCTION WhereY : Byte - возвращают текущие относительные координаты курсора (позицию и строку).
13. PROCEDURE DelLine - удаляет строку окна, в которой находится курсор, все нижние строки автоматически смещаются вверх.
14. PROCEDURE InsLine - вставляет пустую строку перед строкой, в которой находится курсор, все нижние строки автоматически смещаются вниз, и последняя строка окна теряется.
Таким образом, модуль CRT устанавливает режим работы адаптера дисплея, организует вывод в буфер экрана, регулирует яркость свечения символов и т.д. С момента подключения пользователю доступны все содержащиеся в нем стандартные средства. Рассмотрим некоторые из них.

Установка текстового режима - TextMode(Mode:integer); Значение Mode равно 1 (40 / 25) или 3(80 / 25).

Очистка экрана - ClrEol – стирает все символы в строке, начиная с текущей позиции до конца строки;

Управление цветом - Чтобы добавить при выводе эффект мерцания, при установке цвета указывается Blink (16).

7.2 Графика в Турбо Паскале

Экран дисплея ПК представляет собой прямоугольное поле, состоящее из большого количества точек. Дисплей может работать в текстовом и графическом режимах. Но в отличие от текстового режима в графическом режиме имеется возможность изменять цвет каждой точки.

Чтобы сделать процесс графического программирования более эффективным, фирма Borland International разработала специализированную библиотеку Graph (в этом библиотечном модуле содержится 79 графических процедур, функций, различных стандартных констант и типов данных), набор драйверов, позволяющих работать с разными типами монито-

ров, и набор шрифтов для вывода на графический экран текстов разной величины и формы.

Аппаратная поддержка графики ПК обеспечивается двумя основными модулями: видеомонитором и видеоадаптером. Какой бы адаптер ни был установлен на компьютере, мы можем использовать один и тот же набор графических процедур и функций Турбо Паскаля благодаря тому, что их конечная настройка на конкретный адаптер осуществляется автоматически. Эту настройку выполняют графические драйверы.

Запуск и завершение работы в графической системе осуществляется следующим образом:

1. Подключить модуль Graph (библиотеку графических процедур): normal">uses Graph;
2. Установить графический режим:

- описываем переменные, которые определяют графический драйвер и монитор:

var gd, gm: integer;

- задаем команду ПК для самовыбора значений переменных:

gd:=Detect;(значение gm после команды gd:=detect; определяется автоматически)

- инициализируем графический режим:

InitGraph(gd, gm,' указывается путь к драйверу, чем подробнее, тем лучше')

С этого момента все графические средства доступны пользователю.

3. Завершить работу в графической системе: CloseGraph;

Для построения изображений на экране используется система координат. Отсчет начинается от верхнего левого угла экрана, который имеет координаты (0,0). Значение X (столбец) увеличивается слева направо, значение Y (строка) увеличивается сверху вниз. Чтобы строить изображения, необходимо указывать точку начала вывода. В текстовых режимах эту точку указывает курсор, который присутствует на экране. В графических режимах видимого курсора нет, но есть невидимый текущий указатель CP (Current Pointer). Фактически это тот же курсор, но он невидим.

Процедуры модуля Graph

Процедура	**Формат**	**Действие**
SetColor	SetColor (a: word);	Устанавливает цвет, которым будет осуществляться рисование
SetBkColor	SetBkColor (a: word);	Устанавливает цвет фона
SetFillStyle	SetFillStyle (a,b: word); a – стиль закраски, b – цвет	Устанавливает стиль и цвет закраски
SetLineStyle	SetLineStyle (a,b,c:	Устанавливает стиль и толщину

	word); a – стиль линии, b- образец построения линии (может устанавливаться пользователем), с-толщина линии	линии
SetTextStyle	SetTextStyle (a,b,c: word);	Устанавливает шрифт, стиль и размер текста
SetFillPattern	SetFillPattern (Pattern: FillpatternType; Color:word); Pattern- маска	Выбирает шаблон заполнения, определенный пользователем
ClearDivice	ClearDivice	Очищает экран и устанавливает текущий указатель в начало
SetViewPort	SetViewPort (x_1, y_1 , x_2, y_2 : integer, Clip:boolean);	Устанавливает текущее окно для графического вывода
ClearViewPort	ClearViewPort	Очищает окно
PutPixel	PutPixel (a,b,c :integer);	Рисует точку цветом с в (x,y)
Line	Line(x_1, y_1 , x_2,y_2 :integer);	Рисует линию от (x_1, y_1) к (x_2,y_2)
Rectangle	Rectangle (x_1, y_1 , x_2, y_2:integer);	Рисует прямоугольник с диагональю от (x_1, y_1) к (x_2, y_2)
Bar	Bar (x_1, y_1 , x_2, y_2:integer);	Рисует закрашенный прямоугольник
Bar3D	Bar3D (x_1, y_1 , x_2,y_2, d:integer, a:boolean);	Рисует трехмерную полосу (параллелепипед)
Circle	Circle (x,y,r: word);	Рисует окружность радиуса r с центром в точке (x, y)
Arc	Arc(x, y, α, β, R:integer); α, β- начальный и конечный углы в градусах	Рисует дугу из начального угла к конечному, используя (x,y) как центр
Ellipse	Ellipse (x, y, α, β, R_x, R_y: integer); α, β- начальный и конечный углы в градусах	Рисует эллиптическую дугу от начального угла к конечному, используя (x, y) как центр
FillEllipse	FillEllipse (x, y, R_x, R_y:integer); R_x, R_y – вертикальная и	Рисует закрашенный эллипс

	горизонтальная оси	
MoveTo	MoveTo (x, y:integer);	Передвигает текущий указатель в (x, y)
MoveRel	MoveRel(x, y : integer);	Передвигает текущий указатель на заданное расстояние от текущей позиции на x по горизонтали и на y по вертикали
OutText	OutText (text: string);	Выводит текст от текущего указателя
OutTextxy	OutTextxy(x, y: integer, text: string);	Выводит текст из (x, y)
Sector	Sector(x, y, α, β, R_x, R_y: integer); α, β- начальный и конечный углы в градусах	Рисует и заполняет сектор эллипса

Функции модуля Graph

GetBkColor	Возвращает текущий фоновый цвет
GetColor	Возвращает текущий цвет
GetX	Возвращает координату X текущей позиции
GetY	Возвращает координату Y текущей позиции
GetPixel	Возвращает цвет точки в (x, y)

По аналогии с текстовыми режимами графический экран может рассматриваться как одно большое или несколько меньших по размеру окон. После установки окна вся остальная площадь экрана как бы не существует, и весь ввод-вывод осуществляется только через окно. В каждый отдельный момент может быть активным только одно окно. Если окон несколько, за переключение ввода-вывода в нужное окно отвечает программист.

По умолчанию окно занимает весь экран, значения координат его левого верхнего и правого нижнего угла устанавливаются автоматически процедурой инициализации InitGraph.

Если требуется создать окно, следует воспользоваться процедурой SetViewPort (x_1, y_1, x_2, y_2 : integer, Clip:boolean) ; где x_1, y_1 – координаты левого верхнего угла, x_2, y_2 –
координаты правого нижнего угла окна. Параметр Clip определяет, будет ли рисунок отсекаться при выходе за границы окна (Clip:= True) или нет (Clip:=False). После создания окна за точку отсчета принимается верхний левый угол окна, имеющий координаты (0,0).

Координатную систему полного экрана можно восстановить, в частности, с помощью ClearDevice или задав в процедуре установки окна максимально возможные значения:
SetViewPort(0, 0, GetMaxX, GetMaxY, true);

В отличие от текстовых окон графические окна после команды установки фона SetBkColor и очистки с помощью ClearViewPort меняют фон вместе с общим фоном экрана. Поэтому фон (точнее «закраску») графического окна следует устанавливать с помощью процедур SetFillStyle и SetFillPattern.

7.3 Вывод простейших фигур

Вывод точки

Какие бы изображения не выводились на экран, все они построены из точек, теоретически можно создать любое изображение путем построения точек определенного цвета в нужном месте экрана. В библиотеке Graph вывод точки осуществляется процедурой

PutPixel (x, y: integer, color:word);

где x, y: координаты расположения точки, color – цвет.

Возможные значения Color приведены в таблице:

Цветовая шкала

Цвет	Код	Цвет	Код
Black – черный	0	DarkGray – темно-серый	8
Blue – синий	1	LightBlue – голубой	9
Green - зеленый	2	LightGreen – ярко-зеленый	10
Gyan – бирюзовый	3	LightGyan – ярко-бирюзовый	11
Red – красный	4	LightRed – ярко-красный	12
Magenta – малиновый	5	LightMagenta – ярко-малиновый	13
Brown – коричневый	6	Yellow – желтый	14
LightGray – светло-серый	7	White – белый	15

Пример.

PutPixel(320, 240, 4);
PutPixel(320,240, Red);
} - выводит в центре экрана точку красного цвета

Вывод линии

Из точек строятся линии (отрезки прямых). Это можно сделать с помощью процедуры

Line (x_1, y_1 , x_2,y_2 :integer);

где x_1, y_1 – координаты начала, x_2,y_2 - координаты конца линии, например Line(1,1,600,1);

В процедуре Line нет параметра для установки цвета. В этом случае цвет задается процедурой SetColor (цвет: word); где цвет из таблицы 1.

Пример.

```
SetColor(Gyan);
Line(1,1,600,1);
```

Для черчения линий применяются еще две процедуры: LineTo и LineRel. Процедура LineTo (x,y: integer) строит линию из точки текущего положения указателя в точку с координатами x,y. Процедура LineRel (dx,dy: integer) проводит линию от точки текущего расположения указателя (x, y) в точку x+dx, y+dy.

Турбо Паскаль позволяет вычерчивать линии самого различного стиля: тонкие, широкие, штриховые, пунктирные и т.д. Установка стиля производится процедурой SetLineStyle(a,b,c: word), где a устанавливает тип строки; b – образец, c – толщина линии, определяемая константами. Если применяется один из стандартных стилей, то значение b равно 0. Если пользователь хочет активизировать собственный стиль, то значение b =4. В этом случае пользователь сам указывает примитив (образец), из которого строится линия.

Например:

```
SetLineStyle(1,0,1);
Line(15,15, 150,130);
```

или

```
SetLineStyle(UserBitLn,$5555,ThickWidth);
Line(15,15, 150,130);
```

Константа	Значение	Описание
SolidLn	0	Непрерывная линия
DottedLn	1	Линия из точек
CenterLn	2	Линия из точек и тире
DashedLn	3	Штриховая линия
UserBitLn	4	Тип пользователя

Константа	Значение	Описание
NormWidth	1	Нормальная толщина (1 пиксель)
ThickWidth	3	Жирная линия (3 пикселя)

Построение прямоугольников

Для построения прямоугольных фигур имеется несколько процедур. Первая из них – вычерчивание одномерного прямоугольника: Rectangle (x_1, y_1 , x_2, y_2:integer), где x_1, y_1 – координаты левого верхнего угла, x_2, y_2- коорди-

наты правого нижнего угла прямоугольника. Область внутри прямоугольника не закрашена и совпадает по цвету с фоном.

Более эффектные для восприятия прямоугольники можно строить с помощью процедуры Bar (x_1, y_1 , x_2, y_2:integer), которая рисует закрашенный прямоугольник. Цвет закраски устанавливается с помощью SetFillStyle. Еще одна эффектная процедура: Bar3D (x_1, y_1 , x_2,y_2, d: integer, a:boolean) вычерчивает трехмерный закрашенный прямоугольник (параллелепипед). При этом используются тип и цвет закраски, установленные с помощью SetFillStyle. Параметр d представляет собой число пикселей, задающих глубину трехмерного контура. Чаще всего его значение равно четверти ширины прямоугольника (d:= (x_2 - x_1) div 4). Параметр a определяет, строить над прямоугольником вершину (a:=True) или нет (a:=False).

Примеры использования:

1. SetColor(Green);
   Rectangle (200, 100, 250,300);
2. SetFillStyle(1,3);
   Bar(10,10,50,100);
3. SetFillStyle(1,3);
   Bar3D(10,10,50,100,10,True);

Построение многоугольников

Многоугольники можно рисовать самыми различными способами, например с помощью процедуры Line. Однако в Турбо Паскале имеется процедура DrawPoly, которая позволяет строить любые многоугольники линией текущего цвета, стиля и толщины. Она имеет формат DrawPoly(a: word, var PolyPoints)

Параметр PolyPoints является нетипизированным параметром, который содержит координаты каждого пересечения в многоугольнике. Параметр a задает число координат в PolyPoints. Необходимо помнить, что для вычерчивания замкнутой фигуры с N вершинами нужно передать при обращении к процедуре DrawPoly N+1 координату, где координата вершины с номером N будет равна координате вершины с номером 1.

Построение дуг и окружностей

Процедура вычерчивания окружности текущим цветом имеет следующий формат:

Cicrle(x, y, r: word), где x, y – координаты центра окружности, r – ее радиус.

Например, фрагмент программы обеспечит вывод ярко-зеленой окружности с радиусом 50 пикселей и центром в точке (450, 100):

SetColor(LightGreen);
Circle(450, 100, 50);

Дуги можно вычертить с помощью процедуры Arc(x, y: integer, α, β, R:integer), где x, y- центр окружности, α, β- начальный и конечный углы в

градусах, R – радиус. Для задания углов используется полярная система координат.
Цвет для вычерчивания устанавливается процедурой SetColor. В случае α=0 и β=360, вычерчивается полная окружность.

Например, выведем дугу красного цвета от 0 до 90° в уже вычерченной с помощью Circle(450, 100, 50) окружности:

```
SetColor(Red);
Arc(450, 100, 0, 90,50);
```

Для построения эллиптических дуг предназначена процедура Ellipse (x, y: integer, α, β, R_x, R_y: integer), где x, y – центр эллипса, R_x, R_y: горизонтальная и вертикальная оси. В случае α=0 и β=360 вычерчивается полный эллипс. Например, построим голубой эллипс:

```
SetColor(9);
Ellipse(100, 100, 0, 360, 50,50);
```

Фон внутри эллипса совпадает с фоном экрана. Чтобы создать закрашенный эллипс, используется специальная процедура FillEllipse (x, y: integer, R_x, R_y: integer). Закраска эллипса осуществляется с помощью процедуры SetFillStyle (a, b: word), где a – стиль закраски (таблица 4), b – цвет закраски (таблица 1). Например, нарисуем ярко-красный эллипс, заполненный редкими точками зеленого цвета:

```
SetFillStyle(WideDotFill, Green); { установка стиля заполнения}
SetColor(12); {цвет вычерчивания эллипса}
FillEllipse(300, 150, 50, 50);
```

<u>Стандартные стили заполнения</u>

Константа	Значение	Маска
EmptyFill	0	Заполнение цветом фона
SolidFill	1	Заполнение текущим цветом
LineFill	2	Заполнение символами --, цвет – color
LtslashFill	3	Заполнение символами // нормальной толщины, цвет – color
SlashFill	4	Заполнение символами // удвоенной толщины, цвет – color
BkslashFill	5	Заполнение символами \\ удвоенной толщины, цвет – color
LtbkSlahFill	6	Заполнение символами \\ нормальной толщины, цвет – color
HatchFill	7	Заполнение вертикально-горизонтальной штриховкой тонкими линиями, цвет – color
XhatchFill	8	Заполнение штриховкой крест-накрест

		по диагонали «редкими» тонкими линиями, цвет – color
InterLeaveFill	9	Заполнение штриховкой крест-накрест по диагонали «частыми» тонкими линиями, цвет – color
WideDotFill	10	Заполнение «редкими» точками
CloseDotFill	11	Заполнение «частыми» точками
UserFill	12	Заполнение по определенной пользователем маске заполнения, цвет – color

Для построения секторов можно использовать следующие процедуры:
PieSlice (x, y: integer, α, β, R: word), которая рисует и заполняет сектор круга. Координаты x, y – центр окружности, сектор рисуется от начального угла α до конечного угла β, а закрашивание происходит при использовании специальных процедур;
Sector (x, y: integer, α, β, R_x, R_y: word), которая создает и заполняет сектор в эллипсе. Координаты x, y – центр, β, R_x, R_y – горизонтальный и вертикальный радиусы, и сектор вычерчивается от начального угла α до конечного угла β.

Пример использования PieSlice:

```
SetFillStyle(10, 10); {установка стиля}
SetColor(12); {цвет вычерчивания}
PieSlice(100, 100, 0, 90, 50);
```

Пример использования Sector:

```
SetFillStyle(11, 9); {установка стиля}
SetColor(LightMagenta); {цвет вычерчивания}
Sector(300, 150, 180, 135, 60, 70);
```

Вывод текста

Выводимые на экран изображения лучше всего сопровождать пояснительным текстом. В графическом режиме для этого используются процедуры OutText и OutTextXY.

Процедура OutText(Textst: string) выводит строку текста, начиная с текущего положения указателя. Например, OutText(‘нажмите любую клавишу’); Недостаток этой процедуры – нельзя указать произвольную точку начала вывода.

В этом случае удобнее пользоваться процедурой OutTextXY(x, y: integer, Textst: string), где x, y – координаты точки начала вывода текста, Textst – константа или переменная типа String. Например, OutTextXY(60, 100, ‘Нажмите любую клавишу’)

Вывод численных значений

В модуле Graph нет процедур, предназначенных для вывода численных данных. Поэтому для вывода чисел сначала нужно преобразовать их в

строку с помощью процедуры Str, а затем подключить посредством ‘+’ к выводимой строке.
Например: Max:=34.56;

```
            Str(Max : 6 : 2, Smax);{результат преобразования находится в Smax}
            OutTextXY(400, 40, ‘Максимум=’ + Smax);
```

Для удобства преобразование целочисленных и вещественных типов данных в строку лучше осуществлять специализированными пользовательскими функциями IntSt и RealSt:

```
 function IntSt(Int: integer) : string;
 var Buf : string[10];
begin
        Str(Int, Buf);
        IntSt := Buf;
end;
function RealSt(R : real, Dig, Dec : integer) : string;
var Buf: string[20];
begin
        Str(R : Dig : Dec, Buf);
        RealSt := Buf;
end;
```

Эти функции указываются как параметры в процедурах OutText и OutTextXY. Например: x:= 5.295643871;

```
        OutTextXY(20, 20, ‘x=’+RealSt(x,11,9));
```

В результате на экране появится x=5.29564443871

Шрифты

Вывод текста в графическом режиме может осуществляться различными стандартными (таблица 5) и пользовательскими шрифтами. Различают два типа шрифтов: растровые и векторные. Растровый шрифт задается матрицей точек, а векторный – рядом векторов, составляющих символ.

По умолчанию после инициализации графического режима устанавливается растровый шрифт DefaultFont, который, как правило, является шрифтом, используемым драйвером клавиатуры.

Стандартные шрифты

Шрифт	Файл
TriplexFont	Trip.chr
SmallFont	Litt.chr
SansSerifFont	Sans.chr
GothicFont	Goth.chr

Большинство стандартных шрифтом не содержат русских символов. Установить нужный шрифт можно процедурой SetTextStyle(Font,d,c:word), где Font – выбранный шрифт, d – направление (горизонтальное или вертикальное), c – размер выводимых символов. Возможные значения двух первых параметров представлены в таблице. При организации вертикального вывода необходимо учитывать, что если не установить точку начала вывода с помощью MoveTo, то текст начинается с нижней строки экрана и продолжается вверх. Величина символов устанавливается коэффициентом с. Если с=1, то символ строится в матрице 8×8, если с= 2, то матрица 16×16 и т.д. до 10-кратного увеличения.

Например, выведем 2 строки (вертикальную и горизонтальную) шрифтом DefaultFont разной величины:

```
SetTextStyle(0,11); {буквы стандартной величины}
OutTextXY(200,200, 'Вертикальная строка');
SetTextStyle(0,0,2); {размер букв увеличен}
OutTextXY(200,220, 'Горизонтальная строка');
```

Выравнивание текста

В некоторых случаях требуется в пределах одной строки выводить символы выше или ниже друг друга. Выравнивание текста выполняется с помощью процедуры SetTextJustify(Horiz, Vert : word) как по вертикали, так и по горизонтали посредством задания параметров Horiz и Vert (возможные значения в таблице 6).

Параметры выравнивания

Параметр	Значение	Комментарий
Горизонтальное выравнивание		
LeftText	0	Выровнять влево
CenterText	1	Центрировать
RightText	2	Выровнять вправо
Вертикальное выравнивание		
BottomText	0	Переместить вниз
CenterText	1	Центрировать
TopText	2	Переместить вверх

В качестве примера выведем x^2:

```
SetTextJustify(1, 1);
OutTextXY(100,100, 'X');
SetTextJustify(1, 0);
OutTextXY(108,100, '2');
```

Вопросы для самоподготовки:

1. Что такое библиотека CRT в Turbo Pascal и как она используется?
2. Какие директивы модуля CRT вы знаете?

3. Как очищается экран и задается цвет текста и цвет фона?
4. Как устанавливается и отключается звуковой сигнал?
5. Что такое модуль Graph и какие директивы модуля Graph Вы знаете?

Глава 8. Объектно-ориентированное программирование

В основе того или иного языка программирования лежит некоторая руководящая идея, оказывающая существенное влияние на стиль соответствующих программ.

Исторически первой была идея процедурного структурирования программ, в соответствии с которой программист должен был решить, какие именно процедуры он будет использовать в своей программе, а затем выбрать наилучшие алгоритмы для реализации этих процедур. Появление этой идеи было следствием недостаточной изученности алгоритмической стороны вычислительных процессов, столь характерной для ранних программных разработок (сороковые - пятидесятые годы). Типичным примером процедурно-ориентированного языка является Фортран - первый и все еще один из наиболее популярных языков программирования. Последовательное использование идеи процедурного структурирования программ привело к созданию обширных библиотек программирования, содержащих множество сравнительно небольших процедур, из которых, как из кирпичиков, можно строить «здание» программы.

По мере прогресса в области вычислительной математики акцент в программировании стал смещаться с процедур в сторону организации данных. Оказалось, что эффективная разработка сложных программ нуждается в действенных способах контроля правильности использования данных. Контроль должен осуществляться как на стадии компиляции, так и при прогоне программ, в противном случае, как показала практика, резко возрастают трудности создания крупных программных проектов. Отчетливое осознание этой проблемы привело к созданию Алгола-60, а позже - Паскаля, Модулы-2, Си и множества других языков программирования, имеющих более или менее развитые структуры типов данных. Логическим следствием развития этого направления стал модульный подход к разработке программ, характеризующийся стремлением «спрятать» данные и процедуры внутри модуля.

Начиная с языка Симула-67, в программировании наметился новый подход, который получил название объектно-ориентированного программирования (ООП). Его руководящая идея заключается в стремлении связать данные с обрабатывающими эти данные процедурами в единое целое -

объект. Характерной чертой объектов является инкапсуляция (объединение) данных и алгоритмов их обработки, в результате чего и данные, и процедуры во многом теряют самостоятельное значение. Фактически объектно-ориентированное программирование можно рассматривать как модульное программирование нового уровня, когда вместо во многом случайного, механического объединения процедур и данных акцент делается на их смысловую связь.

Какими мощными средствами располагает объектно-ориентированное программирование, наглядно демонстрирует библиотека Turbo Vision, входящая в комплект поставки Турбо Паскаля. В этой главе мы рассмотрим основные идеи ООП и способы их использования.

Следует заметить, что преимущества ООП в полной мере проявляются лишь при разработке достаточно сложных программ. Более того, инкапсуляция придает объектам совершенно особое свойство «самостоятельности», максимальной независимости от остальных частей программы. Правильно сконструированный объект располагает всеми необходимыми данными и процедурами их обработки, чтобы успешно реализовать требуемые от него действия. Попытки использовать ООП для программирования несложных алгоритмов, связанных, например, с расчетными вычислениями по готовым формулам, чаще всего выглядят искусственными нагромождениями ненужных языковых конструкций. Такие программы обычно не нуждаются в структуризации, расчленении алгоритма на ряд относительно независимых частей, их проще и естественнее разрабатывать традиционными способами Паскаля. При разработке сложных диалоговых программ программист вынужден структурировать программу, так как только в этом случае он может рассчитывать на успех: «критической массой» неструктурированных программ является объем в 1000-1200 строк исходного текста - отладка неструктурированных программ большего объема обычно сталкивается с чрезмерными трудностями. Структурирование программы ведет, фактически, к разработке собственной библиотеки программирования - вот в этот момент к Вам на помощь и приходят новые средства ООП.

Таким образом, ООП представляет собой новый этап развития современных концепций построения языков программирования. Здесь получили дальнейшее развитие принципы структурного программирования – структуризация программ и данных, модульность и т.д.

В основе ООП лежит понятие объекта, сочетающего в себе данные и действия над ними. Объект в некотором роде похож на стандартный тип-запись, но включает в себя не только поля данных, но и подпрограммы для обработки этих данных, называемые методами. Таким образом, в объекте сосредоточены его свойства и поведение.

ООП характеризуется тремя основными свойствами: инкапсуляция, наследование и полиморфизм.

Инкапсуляция означает объединение в одном объекте данных и действий над ними. Например, объект – перемещаемый по экрану отрезок, данные – координаты его концов, метод – процедура, обеспечивающая его перемещение.
Наследование позволяет создавать иерархию объектов, начиная с некоторого простого первоначального (предка) и заканчивая более сложными (потомками), включающими в себя свойства предшествующих элементов. Эта иерархия может иметь древовидную структуру. Каждый потомок несет в себе характеристики своего предка (те же данные и методы), а также обладает собственными характеристиками. Например, точка на экране со своими координатами (предок); отрезок, задаваемый координатами двух точек (потомок точки); перемещаемый отрезок, задаваемый координатами концов и процедурой перемещения (потомок неперемещаемого отрезка).
Полиморфизм означает, что для различных родственных объектов можно задать единый класс действий. Затем для каждого конкретного объекта своя подпрограмма, выполняющая это действие непосредственно. Например, перемещение по экрану любой геометрической фигуры. Когда потребуется перемещать конкретную фигуру, то будет выбрана из всего класса соответствующая подпрограмма, т.к. перемещение по экрану точки отличается от перемещения отрезка.
Преимущества ООП:

- Простота введения понятий;
- Сокращение размера программ за счет того, что наследуемые свойства и действия можно не описывать многократно;
- Возможность создания библиотеки объектов;
- Внесение изменений в программу без перекомпиляции уже написанных частей;
- Более четкая локализация свойств и поведения объекта в одном месте;
- Возможность разделения доступа к различным объектам программы.

8.1 Создание объектов

В Турбо Паскале для создания объектов используются три зарезервированных слова: object, constructor, destructor к три стандартные директивы: private, public и virtual.
Зарезервированное слово object используется для описания объекта. Описание объекта должно помещаться в разделе описания типов:
type
MyObject = object
(Поля объекта}

{Методы объекта}
end ;
Если объект порождается от какого-либо родителя, имя родителя указывается в круглых скобках сразу за словом object:

```
type
MyDescendantObject = object(MyObject)
end;
```

Любой объект может иметь сколько угодно потомков, но только одного родителя, что позволяет создавать иерархические деревья наследования объектов.
Для нашей учебной задачи создадим объект-родитель TGraphObject, в рамках которого будут инкапсулированы поля и методы, общие для всех остальных объектов:

```
type
TGraphObj = object
Private {Поля объекта будут скрыты от пользователя}
X,Y: Integer; {Координаты реперной точки}
Color: Word; {Цвет фигуры}
Public {Методы объекта будут доступны пользователю}
Constructor Init(aX,aY: Integer; aColor: Word);
{Создает экземпляр объекта}
Procedure Draw(aColor: Word); Virtual;
{Вычерчивает объект заданным цветом aColor}
Procedure Show;
{Показывает объект - вычерчивает его цветом Color}
Procedure Hide;
{Прячет объект - вычерчивает его цветом фона}
Procedure MoveTo(dX,dY: Integer);
{Перемещает объект в точку с координатами X+dX и Y+dY}
end; {Конец описания объекта TGraphObj}
```

В дальнейшем предполагается создать объекты-потомки от TGraphObj, реализующие все специфические свойства точки, линии, окружности и прямоугольника. Каждый из этих графических объектов будет характеризоваться положением на экране (поля X и Y) и цветом (поле Color). С помощью метода Draw он будет способен отображать себя на экране, а с помощью свойств «показать себя» (метод Show) и «спрятать себя» (метод Hide) сможет перемещаться по экрану (метод MoveTo). Учитывая общность свойств графических объектов, мы объявляем абстрактный объект TGraphObj, который не связан с конкретной графической фигурой. Он объединяет в себе все общие поля и методы реальных фигур и будет служить родителем для других объектов.

Директива Private в описании объекта открывает секцию описания скрытых полей и методов. Перечисленные в этой секции элементы объекта «не видны» программисту, если этот объект он получил в рамках библиотечного TP(/-модуля. Скрываются обычно те поля и методы, к которым программист (в его же интересах!) не должен иметь непосредственного доступа. В нашем примере он не может произвольно менять координаты реперной точки (X.Y), т.к. это не приведет к перемещению объекта. Для изменения полей X и Y предусмотрены входящие в состав объекта методы Init и MoveTo. Скрытые поля и методы доступны в рамках той программной единицы (программы или модуля), где описан соответствующий объект. В дальнейшем предполагается, что программа будет использовать модуль GraphObj с описанием объектов. Скрытые поля будут доступны в модуле GraphObj, но недоступны в использующей его основной программе. Разумеется, в рамках реальной задачи создание скрытых элементов объекта вовсе необязательно. Я ввел их в объект TGraphObj лишь для иллюстрации возможностей ООП.

Директива public отменяет действие директивы private, поэтому все следующие за public элементы объекта доступны в любой программной единице. Директивы private и public могут произвольным образом чередоваться в пределах одного объекта.

Вариант объявления объекта TGraphObj без использования механизма private...public:

```
type
TGraphObj = object
X,Y: Integer;
Color: Word;
Constructor Init(aX,aY: Integer; aColor: Word);
Procedure Draw(aColor: Word); Virtual;
Procedure Show;
Procedure Hide;
Procedure MoveTo(dX,dY: Integer);
end;
```

Описания полей ничем не отличаются от описания обычных переменных. Полями могут быть любые структуры данных, в том числе и другие объекты. Используемые в нашем примере поля X и Y содержат координату реперной (характерной) точки графического объекта, а поле Color - его цвет. Реперная точка характеризует текущее положение графической фигуры на экране и, в принципе, может быть любой ее точкой.

В нашем примере она совпадает с координатами точки в описываемом ниже объекте TPoint, с центром окружности в объекте TCircle, первым концом прямой в объекте TLine и с левым верхним углом прямоугольника в объекте TRect.

Для описания методов в ООП используются традиционные для Паскаля процедуры и функции, а также особый вид процедур - конструкторы и деструкторы. Конструкторы предназначены для создания конкретного экземпляра объекта, ведь объект - это тип данных, т.е. «шаблон», по которому можно создать сколько угодно рабочих экземпляров данных объектного типа (типа TGraphOhj, например). Зарезервированное слово constructor, используемое в заголовке конструктора вместо procedure, предписывает компилятору создать особый код пролога, с помощью которого настраивается так называемая таблица виртуальных методов (см. ниже). Если в объекте нет виртуальных методов, в нем может не быть ни одного конструктора, наоборот, если хотя бы один метод описан как виртуальный (с последующим словом Virtual, см. метод Draw), в состав объекта должен входить хотя бы один конструктор и обращение к конструктору должно предшествовать обращению к любому виртуальному методу.
Типичное действие, реализуемое конструктором, состоит в наполнении объектных полей конкретными значениями. Следует заметить, что разные экземпляры одного и того же объекта отличаются друг от друга только содержимым объектных полей, в то время как каждый из них использует одни и те же объектные методы. В нашем примере конструктор Init объекта TGraphObj получает все необходимые для полного определения экземпляра данные через параметры обращения aX, aY и aColor.
Процедура Draw предназначена для вычерчивания графического объекта. Эта процедура будет реализовываться в потомках объекта TGraphObj по-разному. Например, для визуализации точки следует вызвать процедуру PutPixel, для вычерчивания линии - процедуру Line и т.д. В объекте TGraphObj процедура Draw определена как виртуальная («воображаемая»). Абстрактный объект TGraphObj не предназначен для вывода на экран, однако наличие процедуры Draw в этом объекте говорит о том, что любой потомок TGraphObj должен иметь собственный метод Draw, с помощью которого он может показать себя на экране.
При трансляции объекта, содержащего виртуальные методы, создается так называемая таблица виртуальных методов (ТВМ), количество элементов которой равно количеству виртуальных методов объекта. В этой таблице будут храниться адреса точек входа в каждый виртуальный метод. В нашем примере ТВМ объекта TGraphObj хранит единственный элемент - адрес метода Draw. Первоначально элементы ТВМ не содержат конкретных адресов. Если бы мы создали экземпляр объекта TGraphObj с помощью вызова его конструктора Init, код пролога конструктора поместил бы в ТВМ нужный адрес родительского метода Draw. Далее мы создадим несколько потомков объекта TGraphObj. Каждый из них будет иметь собственный конструктор, с помощью которого ТВМ каждого потомка настраивается так, чтобы ее единственный элемент содержал адрес нужного мето-

да Draw. Такая процедура называется поздним связыванием объекта. Позднее связывание позволяет методам родителя обращаться к виртуальным методам своих потомков и использовать их для реализации специфичных для потомков действий.
Наличие в объекте TGraphObj виртуального метода Draw позволяет легко реализовать три других метода объекта: чтобы показать объект на экране в методе Show, вызывается Draw с цветом aColor, равным значению поля Color, а чтобы спрятать графический объект, в методе Hide вызывается Draw со значением цвета GetBkColor, т.е. с текущим цветом фона.
Рассмотрим реализацию перемещения объекта. Если потомок TGraphObj (например, TLine) хочет переместить себя на экране, он обращается к родительскому методу MoveTo. В этом методе сначала с помощью Hide объект стирается с экрана, а затем с помощью Show показывается в другом месте. Для реализации своих действий и Hide, и Show обращаются к виртуальному методу Draw. Поскольку вызов MoveTo происходит в рамках объекта TLine, используется ТВМ этого объекта и вызывается его метод Draw, вычерчивающий прямую. Если бы перемешалась окружность, ТВМ содержала бы адрес метода Draw объекта TCircle и визуализация-стирание объекта осуществлялась бы с помощью этого метода.
Чтобы описать все свойства объекта, необходимо раскрыть содержимое объектных методов, т.е. описать соответствующие процедуры и функции. Описание методов производится обычным для Паскаля способом в любом месте раздела описаний, но после описания объекта. Например:

```
type
TGraphObj = object
...
end;
Constructor TGraphObj.Init;
begin
X := aX;
Y := aY; Color := aColor
end;
Procedure TGraphObj-Draw;
begin
{Эта процедура в родительском объекте ничего не делает, поэтому экземпляры TGraphObj не способны отображать себя на экране. Чтобы потомки объекта TGraphObj были способны отображать себя, они должны перекрывать этот метод}
end;
Procedure TGraphObj.Show;
begin
Draw(Color)
```

```
end;
Procedure TGraphObj.Hide;
begin
Draw(GetBkColor)
end;
Procedure TGraphObj.MoveTo;
begin
Hide;
X := X+dX;
Y := Y+dY;
Show
end;
```

Отмечу два обстоятельства. Во-первых, при описании методов имя метода дополняется спереди именем объекта, т.е. используется составное имя метода. Это необходимо по той простой причине, что в иерархии родственных объектов любой из методов может быть перекрыт в потомках. Составные имена четко указывают принадлежность конкретной процедуры. Во-вторых, в любом объектном методе можно использовать инкапсулированные поля объекта почти так, как если бы они были определены в качестве глобальных переменных. Например, в конструкторе TGraph.Init переменные в левых частях операторов присваивания представляют собой объектные поля и не должны заново описываться в процедуре. Более того, описание

```
Constructor TGraphObj.Init;
var
X,Y: Integer; {Ошибка!}
Color: Word; {Ошибка!}
begin
end;
```

вызовет сообщение о двойном определении переменных X, Y и Color (в этом и состоит отличие в использовании полей от глобальных переменных: глобальные переменные можно переопределять в процедурах, в то время как объектные поля переопределять нельзя).

Обратите внимание: абстрактный объект TGraphObj не предназначен для вывода на экран, поэтому его метод Draw ничего не делает. Однако методы Hide, Show и MoveTo «знают» формат вызова этого метода и реализуют необходимые действия, обращаясь к реальным методам Draw своих будущих потомков через соответствующие ТВМ. Это и есть полиморфизм объектов.

Создадим простейшего потомка от TGraphObj - объект TPoint, с помощью которого будет визуализироваться и перемещаться точка. Все основные

действия, необходимые для этого, уже есть в объекте TGraphObj, поэтому в объекте TPoint перекрывается единственный метод - Draw.

```
type
TPoint = object(TGraphObj)
Procedure Draw(aColor); Virtual;
end;
Procedure TPoint.Draw;
begin
PutPixel(X,Y,Color) {Показываем цветом Color пиксель с координатами X и Y}
end;
```

В новом объекте TPoint можно использовать любые методы объекта-родителя TGraphObj. Например, вызвать метод MoveTo, чтобы переместить изображение точки на новое место. В этом случае родительский метод TGraphObj.MoveTo будет обращаться к методу TPoint.Draw, чтобы спрятать и затем показать изображение точки. Такой вызов станет доступен после обращения к конструктору Init объекта TPoint, который нужным образом настроит ТВМ объекта. Если вызвать TPoint.Draw до вызова Init, его ТВМ не будет содержать правильного адреса и программа «зависнет».

Чтобы создать объект-линию, необходимо ввести два новых поля для хранения координат второго конца. Дополнительные поля требуется наполнить конкретными значениями, поэтому нужно перекрыть конструктор родительского объекта:

```
type
TLine = object(TGraphObj)
dX,dY: Integer; {Приращения координат второго конца}
Constructor Init(X1,Y1,X2,Y2: Integer; aColor: Word);
Procedure Draw(aColor: Word); Virtual;
end; ,
Constructor TLine.Init;
{Вызывает унаследованный конструктор TGraphObj для инициации полей X, Y и Color. Затем инициирует поля dX и dY}
begin
{Вызываем унаследованный конструктор}
Inherited Init(XI,Yl,aColor);
{Инициируем поля dX и dY}
dX := X2-X1;
dY := Y2-Y1
end;
Procedure Draw;
begin
SetColor(Color);{Устанавливаем цвет Color}
```

Line(X,Y,X+dX,Y+dY){Вычерчиваем линию}
end;

В конструкторе TLine.Init для инициации полей X, Y и Color, унаследованных от родительского объекта, вызывается унаследованный конструктор TGraph.Init, для чего используется зарезервированное слово inherited (англ.- унаследованный):

Inherited Init(XI,Yl,aColor) ;

С таким же успехом мы могли бы использовать и составное имя метода:

TGraphObj.Init(Xl,Yl,aColor);

Для инициации полей dX и dY вычисляется расстояние в пикселах по горизонтали и вертикали от первого конца прямой до ее второго конца. Это позволяет в методе TLine.Draw вычислить координаты второго конца по координатам первого и смещениям dX и dY. В результате простое изменение координат реперной точки X, Y в родительском методе TGraph.MoveTo перемещает всю фигуру по экрану.

Теперь нетрудно реализовать объект TCircle для создания и перемещения окружности:

```
type
TCircle = object(TGraphObj)
R: Integer; {Радиус}
Constructor Init(aX,aY,aR: Integer;
Procedure Draw(aColor: Virtual);
end ;
Constructor TCircle.Init;
begin
Inherited Init(aX,aY,aColor);
R := aR
end ;
aColor: Word)
Procedure TCircle.Draw;
begin
SetColor(aColor); {Устанавливаем цвет Color}
Circle(X,Y,R) {Вычерчиваем окружность}
end;
```

В объекте TRect, с помощью которого создается и перемещается прямоугольник, учтем то обстоятельство, что для задания прямоугольника требуется указать четыре целочисленных параметра, т.е. столько же, сколько для задания линии. Поэтому объект TRect удобнее породить не от TGraphObj, а от TLine, чтобы использовать его конструктор Init:

```
type
TRect = object(TLine)
Procedure Draw(aColor: Word);
```

```
end;
Procedure TRect.Draw;
begin
SetColor(aColor);
Rectangle(X,Y,X+dX,Y+dY) {Вычерчиваем прямоугольник}
end;
```

Чтобы описания графических объектов не мешали созданию основной программы, оформим эти описания в отдельном модуле GraphObj:

```
Unit GraphObj; Interface
{Интерфейсная часть модуля содержит только объявления объектов}
type
TGraphObj = object
...
end;
TPoint = object(TGraphObj)
...
end;
TLine = object(TGraphObj)
...
end;
TCircle = object(TGraphObj)
end;
TRect = object(TLine)
...
end;
Implementation
{Исполняемая часть содержит описания всех объектных методов}
Uses Graph;
Constructor TGraphObj.Init;
...
end.
```

В интерфейсной части модуля приводятся лишь объявления объектов, подобно тому как описываются другие типы данных, объявляемые в модуле доступными для внешних программных единиц. Расшифровка объектных методов помещается в исполняемую часть implementation, как если бы это были описания обычных интерфейсных процедур и функций. При описании методов можно опускать повторное описание в заголовке параметров вызова. Если они все же повторяются, они должны в точности соответствовать ранее объявленным параметрам в описании объекта. Например, заголовок конструктора TGraphObj.Init может быть таким:

```
Constructor TGraphObj.Init;
```

или таким:

```
Constructor TGraphObj.Init(aX,aY: Integer; aColor: Word);
```

8.2 Использование объектов

Идею инкапсуляции полей и алгоритмов можно применить не только к графическим объектам, но и ко всей программе в целом. Ничто не мешает нам создать объект-программу и «научить» его трем основным действиям: инициации (Init), выполнению основной работы (Run) и завершению (Done). На этапе инициации экран переводится в графический режим работы и создаются и отображаются графические объекты (100 экземпляров TPoint и по одному экземпляру TLine, TCircle, TRecf). На этапе Run осуществляется сканирование клавиатуры и перемещение графических объектов. Наконец, на этапе Done экран переводится в текстовый режим и завершается работа всей программы.

Назовем объект-программу именем TGraphApp и разместим его в модуле GraphApp (пока не обращайте внимание на точки, скрывающие содержательную часть модуля -позднее будет представлен его полный текст):

```
Unit GraphApp;
Interface
type
TGraphApp = object
Procedure Init;
Procedure Run;
Destructor Done;
end;
Implementation Procedure TGraphApp.Init;
...
end;
...
end.
```

В этом случае основная программа будет предельно простой:

```
Program Graph_0bjects;
Uses GraphApp;
var
App: TGraphApp;
begin
App.Init;
App.Run;
App.Done
end.
```

В ней мы создаем единственный экземпляр App объекта-программы TGrahpApp и обращаемся к трем его методам.

Создание экземпляра объекта ничуть не отличается от создания экземпляра переменной любого другого типа. Просто в разделе описания переменных мы указываем имя переменной и ее тип:

```
var
App: TGraphApp;
```

Получив это указание, компилятор зарезервирует нужный объем памяти для размещения всех полей объекта TGraphApp. Чтобы обратиться к тому или иному объектному методу или полю, используется составное имя, причем первым указывается не имя объектного типа, а имя соответствующей переменной:

```
App.Init;
App.Run;
App.Done;
```

Переменные объектного типа могут быть статическими или динамическими, т.е. располагаться в сегменте данных (статические) или в куче (динамические). В последнем случае мы могли бы использовать такую программу:

```
Program Graph_0bjects;
Uses GraphApp;
type
PGraphApp = TGraphApp;
var
App: PGraphApp;
begin
App := New(PGraphApp,Init)
App.Run;
App.Done
end;
```

Для инициации динамической переменной App используется вызов функции New. В этом случае первым параметром указывается имя типа инициируемой переменной, а вторым осуществляется вызов метода-конструктора, который, я напомню, нужен для настройки таблицы виртуальных методов. Такой прием (распределение объектов в динамической памяти с одновременной инициацией их ТВМ) характерен для техники ООП. -

Ниже приводится возможный вариант модуля GraphApp для нашей учебной программы:

```
Unit GraphApp;
Interface
Uses GraphObj;
const
NPoints = 100; {Количество точек}
```

```
type
{Объект-программа}
TGraphApp = object
Points: array [1..NPoints] of TPoint; {Массив точек}
Line: TLine; {Линия}
Rect: TRect; {Прямоугольник}
Circ: TCircle; {Окружность}
ActiveObj : Integer; {Активный объект}
Procedure Init; Procedure Run;
Procedure Done; Procedure ShowAll;
Procedure MoveActiveObj (dX,dY: Integer);
end;
Implementation Uses Graph, CRT;
Procedure TGraphApp.Init;
{Инициирует графический режим работы экрана . Создает и отображает NPoints экземпляров объекта TPoint, а также экземпляры объектов TLine, TCircle и TRect}
var
D,R,Err,k: Integer;
begin
{Инициируем графику}
D := Detect; {Режим автоматического определения типа графического адаптера}
InitGraph(D,R, '\tp\bgi') ; {Инициируем графический режим. Текстовая строка должна задавать путь к каталогу с графическими драйверами}
Err := GraphResult; {Проверяем успех инициации графики}
if Err<>0 then
begin
GraphErrorMsg (Err) ;
Halt
end;
{Создаем точки}
for k : = 1 to NPoints do
Points [k] .Init (Random(GetMaxX),Random(GetMaxY),Random(15)+1);
{Создаем другие объекты}
Line. Init (GetMaxX div 3, GetMaxY div 3,2*GetMaxX div 3, 2*GetMaxY div 3,LightRed);
Circ. Init (GetMaxX div 2, GetMaxY div 2, GetMaxY div 5, White);
Rect.Init(2*GetMaxX div 5,2*GetMaxY div 5 , 3*GetMaxX div 5, 3*GetMaxY div 5, Yellow);
ShowAll; {Показываем все графические объекты}
ActiveObj := 1 {Первым перемещаем прямоугольник}
```

```
end ; { TGraphApp .Init}
{----------}
Procedure TGraphApp .Run ;
{Выбирает объект с помощью Tab и перемещает его по экрану}
var
Stop: Boolean; {Признак нажатия Esc}
const
D = 5; {Шаг смещения фигур}
begin
Stop := False;
{Цикл опроса клавиатуры}
repeat
case ReadKey of {Читаем код нажатой клавиши}
#27: Stop := True; {Нажата Esc}
#9:begin {Нажата Tab}
inc(ActiveObj);
if ActiveObj>3 then
ActiveObj := 3
end;
#0: case ReadKey of
#71:MoveActiveObj(-D,-D); {Влево и вверх}
#72:MoveActiveObj( 0,-D); {Вверх}
#73:MoveActiveObj( D,-D); {Вправо и вверх}
#75:MoveActiveObj(-D, 0); {Влево}
#77:MoveActiveObj( D, 0); {Вправо}
#79:MoveActiveObj(-D, D); {Влево и вниз}
#80:MoveActiveObj( 0, D); {Вниз}
#81:MoveActiveObj( D, D); {Вправо и вниз}
end
end;
ShowAll;
Until Stop
end; {TGraphApp. Run}
{----------}
Destructor TGraphApp . Done ;
{Закрывает графический режим}
begin
CloseGraph
end; {TGraphApp. Done}
Procedure TGraphApp . ShowAll ;
{Показывает все графические объекты}
var
```

```
k: Integer;
begin
for k := 1 to NPoints do Points [k] . Show;
Line. Show;
Rect . Show;
Circ.Show
end;
{----------}
Procedure TGraphApp.MoveActiveObj;
{Перемещает активный графический объект}
begin
case ActiveObj of
1: Rect.MoveTo(dX,dY);
2: Circ.MoveTo(dX,dY);
3: Line.MoveTo(dX,dY)
end
end;
end.
```

В реализации объекта TGraphApp используется деструктор Done. Следует иметь в виду, что в отличие от конструктора, осуществляющего настройку ТВМ, деструктор не связан с какими-то специфичными действиями: для компилятора слова destructor и procedure - синонимы. Введение в ООП деструкторов носит, в основном, стилистическую направленность - просто процедуру, разрушающую экземпляр объекта, принято называть деструктором. В реальной практике ООП с деструкторами обычно связывают процедуры, которые не только прекращают работу с объектом, но и освобождают выделенную для него динамическую память. И хотя в нашем примере деструктор Done не освобождает кучу, я решил использовать общепринятую стилистику и заодно обсудить с Вами последнее еще не рассмотренное зарезервированное слово технологии ООП.

В заключении следует сказать, что формалистика ООП в рамках реализации этой технологии в Турбо Паскале предельно проста и лаконична. Согласитесь, что введение лишь шести зарезервированных слов, из которых действительно необходимыми являются три (object, constructor и virtual), весьма небольшая плата за мощный инструмент создания современного программного обеспечения.

Вопросы для самоподготовки.

1. Что такое объект?
2. Что такое методы?
3. Привести пример работы с объектами.
4. Перечислить основные свойства объектно-ориентированного программирования.

Раздел 2. Delphi

Глава 1. Знакомство со средой Delphi

1.1 Общие представления

В России Borland Delphi появляется в конце 1995 г. и сразу же завоевывает широкую популярность. Новые версии выходят практически каждый год. В них реализуются все новые мастера, компоненты и технологии программирования.

Действительно, процесс разработки в Delphi предельно упрощен. В первую очередь это относится к созданию интерфейса, на который уходит 80% времени разработки программы. Вы просто помещаете нужные компоненты на поверхность Windows-окна (в Delphi оно называется формой) и настраиваете их свойства с помощью специального инструмента (Object Inspector). С его помощью можно связать события этих компонентов (нажатие на кнопку, выбор мышью элемента в списке и т.д.) с кодом его обработки - и вот простое приложение готово. Причем разработчик получает в свое распоряжение мощные средства отладки (вплоть до пошагового выполнения команд процессора), удобную контекстную справочную систему, средства коллективной работы над проектом и.т.д. Можно создавать распределенные приложения на базе COM и CORBA, Интернет- и intranet-приложения, используя для доступа к данным Borland DataBase Engine, ODBC-драйверы или Microsoft ADO. Появившаяся, начиная с Delphi 3, поддержка многозвенной технологии (multi-tiered) доступа к данным позволяет создавать масштабируемые приложения (относительно слабо зависящие от сервера БД) за счет перенесения методов обработки информации (бизнес-правил) на среднее звено.

Как уже говорилось ранее, в Delphi используется язык Object Pascal, который постоянно расширяется и дополняется Borland. Язык в полной мере поддерживает все требования, предъявляемые к объектно-ориентированному языку программирования. Как и положено строго типизированному языку, классы поддерживают только простое наследование, но зато интерфейсы могут иметь сразу несколько предков. К числу особенностей языка следует отнести поддержку обработки исключительных ситуаций (exceptions), а также перегрузку методов и подпрограмм (overload) в стиле C++. Также имеются открытые массивы, варианты и вариантные массивы, позволяющие размещать в памяти все, что душе угодно и смешивать типы данных.

Вы можете создавать свои собственные компоненты, импортировать OCX-компоненты, создавать шаблоны проектов и «мастеров», создающих «заготовки» проектов. Кроме того, Delphi предоставляет разработчику интер-

фейс для связи приложений (или внешних программ) с интегрированной оболочкой Delphi (IDE).
Таким образом, вы можете использовать Delphi для создания как самых простых приложений, на разработку которых требуется 2-3 часа, так и серьезных корпоративных проектов, предназначенных для работы десятков и сотен пользователей. Причем для этого можно использовать самые последние веяния в мире компьютерных технологий с минимальными затратами времени и сил.

1.2 Основные отличия различных версий Delphi

Версия 1
Первая версия появилась в мае 1995 г., когда еще не существовала Windows 95 (но, тем не менее, существовала Windows NT). Это единственная версия, работающая под управлением 16-разрядной Windows 3.1 (3.11). В ней впервые была опробована новая модель объектов, позаимствованная из различных объектно-ориентированных языков, и главным образом, из языка C++. Эта модель оказалась настолько революционной, что существовавшие в то время в поздних версиях Turbo Pascal объекты стали не нужны (их возможности полностью поглотила новая модель), а сама новая модель получила название классов.
Классы активно используют динамическую память, в связи с чем несколько изменилась нотация языка, а сам язык был назван Object Pascal. По сравнению с Turbo Pascal в него были внесены существенные дополнения и изменения, в том:
-введены открытые массивы и их конструкторы для передачи в подпрограммы массивов переменной длины;
-введена внутренняя для функции переменная Result и разрешено игнорировать возвращаемый функцией результат;
-сняты ограничения на тип возвращаемого функцией результата (этот тип может быть любым, за исключением объектов старого стиля и файлов);
-введен механизм обработки исключительных ситуаций.
Библиотека компонентов 1-й версии с самого начала показала основную направленность вновь разработанной системы: программирование баз данных. С этой целью первая и все последующие версии Delphi снабжаются специальным инструментом доступа к данным -BDE (Borland Database Engine - машина баз данных корпорации Borland), а также сервером баз данных InterBase (этот сервер производится филиалом Borland - компанией InterBase Software Corporation) и соответствующими средствами конфигурации сервера, его контроля и связи с ним.

Палитра компонентов первой версии состоит из 9 страниц и содержит 79 компонентов. В качестве дополнительных утилит поставлялись система генерации отчетов.

Версия 2

Вторая и все последующие версии Delphi предназначены для работы под управлением 32-разрядных ОС Windows 95/98/2000/NT (Windows 32). В нее были внесены многочисленные изменения, связанные с переходом на качественно новую ОС, в том числе:

-введена поддержка 16-битных (“широких”) символов и составленных из них строк;

-введен новый формат строк “обычных” 8-битных символов произвольной длины;

-введены новые типы данных - variant и currency;

-введен механизм фильтрации в табличные наборы данных TTable.

Начиная с версии 2, Delphi поставляется в нескольких комплектациях, отличающихся набором инструментальных средств и компонентов. Во 2-й версии таких комплектов было 3: Desktop, Professional и Client/Server Suite. На 12 страницах галереи компонентов расположены 114 стандартных компонентов. В том числе на странице QReport размещены удобные компоненты для создания отчетов по хранящимся в базах данным. Эти компоненты оказались значительно эффективнее специальной утилиты ReportSmith, в связи с чем эта утилита не входит в поставку версии 3 и выше.

Версия 3

Основные новшества этой версии:

-использование механизма пакетов для облегчения распространения и повторного использования компонентов;

-улучшенные свойства редактора кода: построение фрагментов кода по образцу; оперативная подсказка о типе и количестве формальных параметров при обращениях к подпрограммам; оперативный просмотр в режиме отладки содержимого полей, свойств и переменных с помощью указателя мыши;

-поддержка шаблонов компонентов;

-поддержка технологий COM, ActiveX, OLEnterprice и (частично) CORBA.

Третья версия поставлялась в 4 комплектациях: Standard, Professional, Client/Server Suite и Enterprise.

На 13 страницах галереи компонентов размещены 148 стандартных компонента. В модификации 3.5 введены дополнительные компоненты для реализации многозвенных баз данных.

Версия 4

Появившаяся в июле 1998 г. 4-я версия Delphi быстро завоевала широкую популярность как своими расширенными языковыми возможностями, так

и специальной поддержкой многозвенных баз данных и распределенных вычислений.
К основным нововведениям этой версии относятся:
-изменения в языке: динамические массивы, перегружаемые методы, умалчиваемые параметры подпрограмм, новые типы int64, Real48 и Real как аналог Double;
-изменения в кодовом редакторе: автоматическое завершение кодовых заголовков свойств и методов; свойства браузера для поиска исходного кода; простой переход от заголовка метода к его реализации и обратно;
-технология “причаливания” инструментальных панелей Drag&Dock;
-механизм “действий” Action для унификации внешнего вида и поведения одинаковых по функциональному назначению интерфейсных элементов;
-улучшенная поддержка многозвенной архитектуры БД и распределенных вычислений.
На 14 страницах палитры компонентов размещены 182 стандартных компонента.
Версия 5
В июле 1999 г. вышла пятая по счету версия Delphi, основная особенность которой - попытка заменить громоздкий и не всегда быстрый механизм доступа к данным BDE, который традиционно использовался во всех предыдущих версиях, альтернативными механизмами.
Для этого, во-первых, в Delphi 5 включена поддержка технологии ADO (ActiveX Data Objects - объекты данных, построенные как объекты ActiveX), которая усиленно развивается корпорацией Microsoft.
Во-вторых, сотрудники Borland и его подразделения InterBase Software Corporation разработали серию компонентов облегченного доступа к данным, хранящимся в таблицах сервера InterBase v.5.5 и выше (страница InterBase палитры компонентов). Эти компоненты также не требуют BDE и, таким образом, создают “облегченное” клиентское место.
Менее значительные изменения, внесенные в версию 5:
-включен эксперт создания и настройки произвольных модулей данных с расширенными возможностями представления взаимосвязи данных;
-улучшена технология MIDAS: для более гибкой работы с Microsoft Transaction Server -введен повторно-входимый (stateless) брокер данных; упрощен процесс разработки интранет-приложений за счет компонентов новой страницы InternetExpress;
значительные изменения внесены в интегрированную среду разработчика ИСР; в том числе:
-для улучшения координации коллективной работы над одним проектом введен новый инструмент - список To-Do;
-программист теперь может при желании использовать несколько вариантов настройки основных окон Delphi - например, для режима кодирования

на экране могут не присутствовать отладочные окна, которые, наоборот, могут понадобиться в отладочном режиме; нужный вариант настройки легко выбирается новыми интерфейсными элементами в главном окне Delphi;
-введены фильтрующие свойства в окне Инспектора Объектов, упрощающие выбор нужного свойства;
-опции Инспектора Объектов теперь могут снабжаться небольшими пиктограммами, облегчающими правильный выбор нужной опции (например, радом с названием каждого цвета показывается небольшой прямоугольник, заполненный этим цветом, рядом с названием курсора - его вид и т. п.);
-существенно усилены возможности встроенного отладчика: точки отладочного останова можно группировать и сделать доступной или недоступной сразу группу точек; с каждой точкой останова можно связать одно или несколько действий, которые будут выполняться при достижении этой точки; с помощью команды Run | Attach to process можно отлаживать процесс, запущенный в другом экземпляре ИСР (эта возможность существенно упрощает отладку многозвенных приложений); с помощью выбора Run | Run Until Return в главном меню можно продолжить пошаговую отладку после завершения текущей подпрограммы и т. д.;
-введены дополнительные возможности в Менеджер Проекта, упрощающий координацию многих разработчиков в рамках единого проекта;
-создан механизм Менеджера Трансляций, облегчающий разработку многоязычных программ в рамках единого проекта;
-внесены изменения в кодовый редактор, позволяющий настраивать используемые в нем “горячие” клавиши;
-несколько переработана встроенная справочная служба;
-внесены многочисленные изменения и дополнения в галереи компонентов (в новой версии наиболее мощной комплектации Enterprise на 20 страницах расположены 218 стандартных компонентов). Версия поставляется в 3 комплектациях: Standard, Professional и Enterprise. Состав утилит, входящих в комплектацию Enterprise, полностью соответствует утилитам комплектации Client/Server Suite предыдущей версии.

<u>Версия 6</u>

Версия 6 вышла в мае 2001 г. 6-я версия Delphi имеет уникальную особенность: она способна создавать так называемые, межплатформенные приложения, т. е. программы, которые одинаково успешно могут работать как под управлением Windows 32, так и под Linux.

Две другие особенности Delphi 6: в ней сделаны дальнейшие шаги для поддержки Web-программирования (архитектура websnap) и разработаны драйверы и компоненты для максимально быстрой связи клиентских мест с некоторыми популярными промышленными серверами баз данных без BDE (компоненты страницы dbExpress).

Технология dbExpress поддерживает непосредственный доступ к таким популярным серверам БД, как MySQL, Oracle, DB2. Таким образом, основной идей Delphi 6 является обеспечение перехода от дорогих патентованных решений корпорации Microsoft к бесплатным (или почти бесплатным) решениям на базе Linux.
На 27 страницах палитры компонентов размещены 387 компонентов.
Версия 7
Появился целый ряд новых компонентов. Во-первых, новый генератор отчетов — Rave. Помимо удобств, которые обеспечивает визуальный редактор, поддерживается вывод во все самые распространенные форматы: обычный текст, HTML, RTF и PDF.
Теперь с помощью встроенных средств Delphi можно создавать приложения, меню и панели инструментов которых даже в Windows 98 будут выглядеть в духе WinXP.
Среда разработки изменилась сравнительно мало — разве что появилось автозавершение кода для HTML, WML, XHTML и XSL.
Интернет остается основной движущей силой прогресса, поэтому закономерно появились новые средства и инструменты, упрощающие работу с вебом. В Delphi 7 значительно расширен список серверов и клиентов (Internet Direct — Indy), так что теперь создать свой собственный «мини-Апач» (а заодно и FTP, и почтовый сервер с поддержкой IMAP и POP, и серверы IRC, Telnet, Whois и т.д.) можно очень быстро. Кроме того, добавлена поддержка одного из самых популярных веб-серверов — Apache 2.0.
Еще одно новшество — мощное средство для создания серверных приложений IntraWeb. При этом серверная часть может быть не только самостоятельным приложением (в виде Windows service), но и модулем для Apache или IIS. Клиентом же выступает браузер — заявлено о поддержке IE, Netscape и Mozilla.
Добавился целый ряд серьезных инструментов для разработчиков. Во-первых, появился UDDI-браузер (Universal Description, Discovery and Integration). Во-вторых, в состав дистрибутива входит ModelMaker, инструмент для разработки, настройки и обслуживания интерфейсов и классов. В-третьих, расширена поддержка XML.
В четвертых, существует поддержка манифеста для Windows XP, так что теперь любое приложение, созданное с использованием Delphi 7, автоматически сможет менять внешний вид через встроенный в WinXP механизм.
Версия 8
В новую версию Delphi интегрированы базовые средства высокоуровневого моделирования приложения. Контекст отладки - в основном рабочем пространстве остаются лишь необходимые окна
Delphi 8 -- это результат полугодового эксперимента. Только теперь в его состав входит уже полноценный компилятор, а сам пакет значительно

трансформировался, для того чтобы наилучшим образом соответствовать современным требованиям к продуктам такого рода, и ориентирован исключительно на разработку .NET-приложений.
Borland включила в каждую поставку Delphi 8 полный вариант Delphi 7. Теперь владелец данного пакета свободен в выборе платформы и может совершать переход от Win32 к .NET постепенно.
Delphi 8 -- это не только удобный и эффективный RAD-пакет, вместе с ним в распоряжение пользователя попадает целая серия мощных и уникальных инструментов.
Delphi 8 доступен в трех редакциях -- Architect, Enterprise и Professional. Architect-издание содержит все средства и функции, которые обеспечивает пакет; владельцы Enterprise-версии будут лишены возможности моделировать приложения с применением Borland LiveSource и архитектуры MDA, совершать тонкую настройку производительности приложения с помощью полного варианта мощного профайлера Optimizeit для .NET; пользователям Delphi 8 Professional, помимо вышеназванных инструментов, недоступны функции интеграции с различными серверами контроля версий (кроме CaliberRM) и некоторые компоненты для работы с базами данных.

Вопросы для самоподготовки:

1. В чем отличия Pascal и Delphi?
2. Как называется язык программирования, интегрированный в среду Delphi?
3. Какие версии Delphi поддерживают работу с базами данных?
4. Какие версии Delphi поддерживают работу в сетях Интернет?

Глава 2. Основы визуального программирования

2.1 Знакомство с компонентами

Среда Delphi включает в себя полный набор визуальных инструментов для скоростной разработки приложений (RAD - rapid application development), поддерживающей разработку пользовательского интерфейса и подключение к корпоративным базам данных. VCL - библиотека визуальных компонент, включает в себя стандартные объекты построения пользовательского интерфейса, объекты управления данными, графические объекты, объекты мультимедиа, диалоги и объекты управления файлами, управление DDE и OLE.
После запуска Delphi в верхнем окне горизонтально располагаются иконки палитры компонент. Если курсор задерживается на одной из иконок, под ней в желтом прямоугольнике появляется подсказка

Из этой палитры компонент вы можете выбирать компоненты, из которых можно строить приложения. Компоненты включают в себя как визуальные, так и логические компоненты. Такие вещи, как кнопки, поля редактирования - это визуальные компоненты; а таблицы, отчеты - это логические.

Поскольку в Delphi вы визуальным образом строите свою программу, все эти компоненты имеют свое графическое представление в поле форм для того, чтобы можно было бы ими соответствующим образом оперировать. Но для работающей программы видимыми остаются только визуальные компоненты. Компоненты сгруппированы на страницах палитры по своим функциям. К примеру, компоненты, представляющие Windows "common dialogs" все размещены на странице палитры с названием "Dialogs".

Delphi позволяет разработчикам настроить среду для максимального удобства. Вы можете легко изменить палитру компонент, инструментальную линейку, а также настраивать выделение синтаксиса цветом.

Заметим, что в Delphi вы можете определить свою группу компонент и разместить ее на странице палитры, а если возникнет необходимость, перегруппировать компоненты или удалить неиспользуемые.

Структура среды программирования.

Внешний вид среды программирования Delphi отличается от многих других из тех, что можно увидеть в Windows. К примеру, Borland Pascal for Windows 7.0, Borland C++ 4.0, Word for Windows, Program Manager - это все MDI приложения и выглядят по-другому, чем Delphi. MDI (Multiple Document Interface) - определяет особый способ управления нескольких дочерних окон внутри одного большого окна.

Среда Delphi же следует другой спецификации, называемой Single Document Interface (SDI), и состоит из нескольких отдельно расположенных окон. Это было сделано из-за того, что SDI близок к той модели приложений, что используется в Windows 95.

Если нужно переключиться на другое приложение, то просто щелкните мышкой на системную кнопку минимизации Delphi. Вместе с главным окном свернутся все остальные окна среды программирования, освободив место для работы других программ.

Главные составные части среды программирования

Ниже перечислены основные составные части Delphi:

1. Дизайнер Форм (Form Designer)
2. Окно Редактора Исходного Текста (Editor Window)
3. Палитра Компонент (Component Palette)
4. Инспектор Объектов (Object Inspector)
5. Справочник (On-line help)

Есть, конечно, и другие важные составляющие Delphi, вроде линейки инструментов, системного меню и многие другие, нужные для точной настройки программы и среды программирования.

Программисты на Delphi проводят большинство времени переключаясь между Дизайнером Форм и Окном Редактора Исходного Текста (которое для краткости называют Редактор). Дизайнер Форм показан на рис.1, окно Редактора - на рис.2.

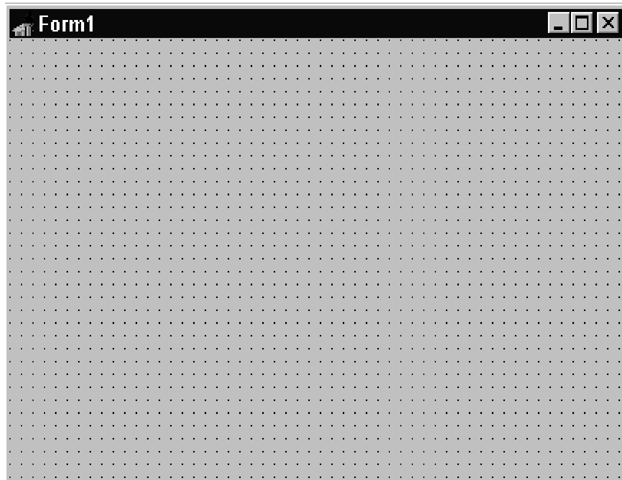


Рис.1.

Рис.2.

Дизайнер Форм первоначально состоит из одного пустого окна, которое Вы заполняете всевозможными объектами, выбранными на Палитре Компонент.

Несмотря на всю важность Дизайнера Форм, местом, где программисты проводят основное время, является Редактор.

Палитра Компонент (см. рис.3) позволяет Вам выбрать нужные объекты для размещения их на Дизайнере Форм. Для использования Палитры Компонент просто первый раз щелкните мышкой на один из объектов и потом второй раз - на Дизайнере Форм. Выбранный Вами объект появится на проектируемом окне и им можно манипулировать с помощью мыши.

Палитра Компонент использует постраничную группировку объектов. Внизу Палитры находится набор закладок - Standard, Additional, Dialogs и т.д. Если Вы щелкнете мышью на одну из закладок, то Вы можете перейти на следующую страницу Палитры Компонент. Принцип разбиения на страницы широко используется в среде программирования Delphi и его легко можно использовать в своей программе. (На странице Additional есть компоненты для организации страниц с закладками сверху и снизу).

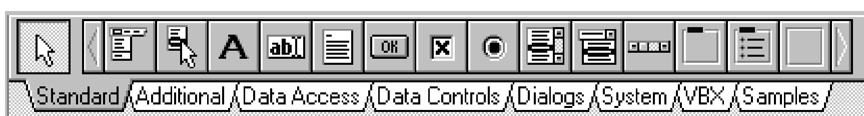
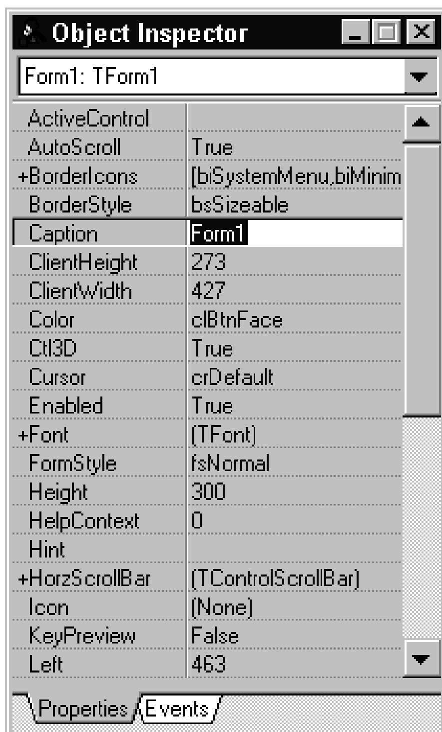


Рис.3: Палитра Компонент - место, где Вы выбираете объекты, которые будут помещены на вашу форму.

Слева от Дизайнера Форм Вы можете видеть Инспектор Объектов (рис.4). Заметьте, что информация в Инспекторе Объектов меняется в зависимости от объекта, выбранного на форме. Важно понять, что каждый компонент является настоящим объектом, и Вы можете менять его вид и поведение с помощью Инспектора Объектов.

Инспектор Объектов состоит из двух страниц, каждую из которых можно использовать для определения поведения данного компонента. Первая страница - это список свойств, вторая - список событий. Если нужно изменить что-нибудь, связанное с определенным компонентом, то Вы обычно делаете это в Инспекторе Объектов. К примеру, Вы можете изменить имя и размер компонента TLabel, изменяя свойства Caption, Left, Top, Height, и Width.



Вы можете использовать закладки внизу Инспектора Объектов для переключения между страницами свойств и событий. Страница событий связана с Редактором; если Вы дважды щелкнете мышкой на правую сторону какого-нибудь пункта, то соответствующий данному событию код автоматически запишется в Редактор, сам Редактор немедленно получит фокус, и Вы сразу же имеете возможность добавить код обработчика данного события.

Последняя важная часть среды Delphi - Справочник (on-line help). Для доступа к этому инструменту нужно просто выбрать в системном меню пункт Help и затем Contents. На экране появится Справочник.

Справочник является контекстно-зависимым; при нажатии клавиши F1, Вы получите подсказку, соответствующую текущей ситуации. Например, находясь в Инспекторе Объектов, выберите какое-нибудь свойство и нажмите F1 - Вы получите справку о назначении данного свойства.

Дополнительные элементы

В данном разделе внимание фокусируется на трех инструментах, которые можно воспринимать как вспомогательные для среды программирования:

- Меню (Menu System)
- Панель с кнопками для быстрого доступа (SpeedBar)
- Редактор картинок (Image Editor)

Меню предоставляет быстрый и гибкий интерфейс к среде Delphi, потому что может управляться по набору “горячих клавиш”. Это удобно еще и потому, что здесь используются слова или короткие фразы, более точные и

понятные, нежели иконки или пиктограммы. Вы можете использовать меню для выполнения широкого круга задач; скорее всего, для наиболее общих задач вроде открытия и закрытия файлов, управления отладчиком или настройкой среды программирования.
SpeedBar находится непосредственно под меню, слева от Палитры Компонент (рис.5). SpeedBar выполняет много из того, что можно сделать через меню. Если задержать мышь над любой из иконок на SpeedBar, то Вы увидите, что появится подсказка, объясняющая назначение данной иконки.

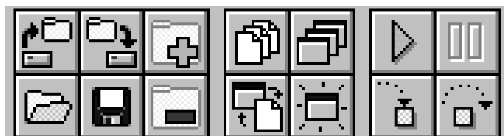


Рис.5: SpeedBar находится слева от Палитры Компонент.
Редактор Картинок, работает аналогично программе Paintbrush из Windows. Вы можете получить доступ к этому модулю, выбрав пункт меню Tools | Image Editor.
А теперь нужно рассмотреть те элементы, которые программист на Delphi использует в повседневной жизни.
Инструментальные средства
В дополнение к инструментам, обсуждавшимся выше, существуют пять средств, поставляемых вместе с Delphi. Эти инструментальные средства:

- Встроенный отладчик
- Внешний отладчик (поставляется отдельно)
- Компилятор командной строки
- WinSight
- WinSpector

Данные инструменты собраны в отдельную категорию потому, что они играют достаточно абстрактную техническую роль в программировании.
Отладчик позволяет пройти пошагово по исходному тексту программы, выполняя по одной строке за раз, и открыть просмотровое окно (Watch), в котором будут отражаться текущие значения переменных программы.
Встроенный отладчик, который наиболее важен из пяти вышеперечисленных инструментов, работает в том же окне, что и Редактор. Внешний отладчик делает все, что делает встроенный и кое-что еще. Он более быстр и мощен, чем встроенный. Однако он не так удобен в использовании, главным образом из-за необходимости покидать среду Delphi.
Теперь давайте поговорим о компиляторах. Внешний компилятор, называется DCC.EXE, полезен, в основном, если Вы хотите скомпилировать приложение перед отладкой его во внешнем отладчике. Большинство программистов, наверняка, посчитают, то гораздо проще компилировать в среде Delphi, нежели пытаться создать программу из командной строки.

Однако, всегда найдется несколько оригиналов, которые будут чувствовать себя счастливее, используя компилятор командной строки. Но это факт - возможно создать и откомпилировать программу на Delphi используя только DCC.EXE и еще одну программу CONVERT.EXE, которая поможет создать формы. Однако, данный подход неудобен для большинства программистов.
WinSight и WinSpector интересны преимущественно для опытных программистов в Windows. Это не значит, что начинающий не должен их запускать и экспериментировать с ними по своему усмотрению. Но эти инструменты вторичны и используются для узких технических целей.
Из этих двух инструментов WinSight определенно более полезен. Основная его функция - позволить Вам наблюдать за системой сообщений Windows. Хотя Delphi делает много для того, чтобы спрятать сложные детали данной системы сообщений от неопытных пользователей, тем не менее Windows является операционной системой, управляемой событиями. Почти все главные и второстепенные события в среде Windows принимают форму сообщений, которые рассылаются с большой интенсивностью среди различными окнами на экране. Delphi дает Вам полный доступ к сообщениям Windows и позволяет отвечать на них, как только будет нужно. В результате, опытным пользователям WinSight становится просто необходим.
WinSpector сохраняет запись о текущем состоянии машины в текстовый файл; Вы можете просмотреть этот файл для того, чтобы узнать, что неправильно идет в программе. Данный инструмент полезен, когда программа находится в опытной эксплуатации - можно получить важную информацию при крушении системы.
Основная палитра компонентов Delphi имеет двенадцать (и более) страниц.

Стандартные компоненты

Для дальнейшего знакомства со средой программирования Delphi потребуется рассказать о составе первой страницы Палитры Компонент.
На первой странице Палитры Компонент (**Standard**) размещены 14 объектов, важных для использования (рис.6). Большинство компонентов на этой странице являются аналогами экранных элементов самой Windows. Меню, кнопки, полосы прокрутки — здесь есть все. Но компоненты Delphi обладают также некоторыми удобными дополнительными встроенными возможностям.
Набор и порядок компонент на каждой странице являются конфигурируемыми. Так, Вы можете добавить к имеющимся компонентам новые, изменить их количество и порядок.

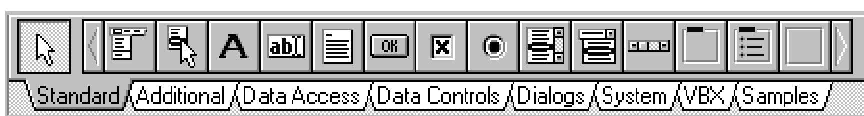


Рис.6: Компоненты, расположенные на первой странице Палитры.

Стандартные компоненты Delphi перечислены ниже с некоторыми комментариями по их применению. При изучении данных компонент было бы полезно иметь под рукой компьютер с тем, чтобы посмотреть, как они работают и как ими манипулировать.

· **TMainMenu** позволяет Вам поместить главное меню в программу. При помещении TMainMenu на форму это выглядит, как просто иконка. Иконки данного типа называют "невидимыми компонентом", поскольку они невидимы во время выполнения программы. Создание меню включает три шага: (1) помещение TMainMenu на форму, (2) вызов Дизайнера Меню через свойство Items в Инспекторе Объектов, (3) определение пунктов меню в Дизайнере Меню.

· **TPopupMenu** позволяет создавать всплывающие меню. Этот тип меню появляется по щелчку правой кнопки мыши.

· **TLabel** служит для отображения текста на экране. Вы можете изменить шрифт и цвет метки, если дважды щелкнете на свойство Font в Инспекторе Объектов. Вы увидите, что это легко сделать и во время выполнения программы, написав всего одну строчку кода.

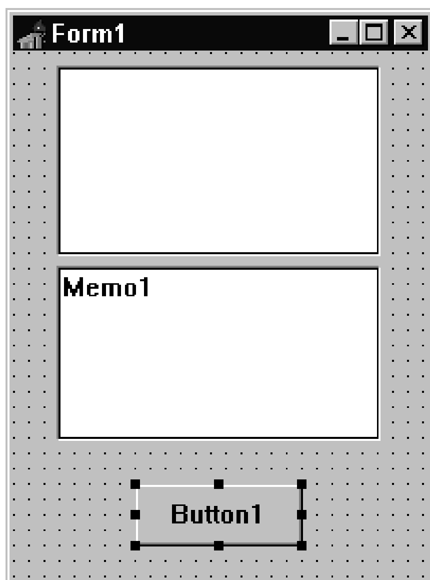
· **TEdit** - стандартный управляющий элемент Windows для ввода. Он может быть использован для отображения короткого фрагмента текста и позволяет пользователю вводить текст во время выполнения программы.

· **TMemo -** иная форма TEdit. Подразумевает работу с большими текстами. TMemo может переносить слова, сохранять в Clipboard фрагменты текста и восстанавливать их, и другие основные функции редактора. TMemo имеет ограничения на объем текста в 32Кб, это составляет 10-20 страниц. (Есть VBX и “родные” компоненты Delphi, где этот предел снят).

· **TButton** позволяет выполнить какие-либо действия при нажатии кнопки во время выполнения программы. В Delphi все делается очень просто. Поместив TButton на форму, Вы по двойному щелчку можете создать заготовку обработчика события нажатия кнопки. Далее нужно заполнить заготовку кодом (подчеркнуто то, что нужно написать вручную):

```
procedure TForm1.Button1Click(Sender: TObject);
begin
MessageDlg('Are you there?',mtConfirmation,mbYesNoCancel,0);
end;
```

· **TCheckBox** отображает строку текста с маленьким окошком рядом. В окошке можно поставить отметку, которая означает, что что-то выбрано. Например, если посмотреть окно диалога настроек компилятора (пункт меню Options | Project, страница Compiler), то можно увидеть, что оно состоит преимущественно из CheckBox’ов.



· **TRadioButton** позволяет выбрать только одну опцию из нескольких. Если Вы опять откроете диалог Options | Project и выберете страницу Linker Options, то Вы можете видеть, что секции Map file и Link buffer file состоят из наборов RadioButton.

· **TListBox** нужен для показа прокручиваемого списка. Классический пример ListBox'а в среде Windows - выбор файла из списка в пункте меню File | Open многих приложений. Названия файлов или директорий и находятся в ListBox'е.

· **TComboBox** во многом напоминает ListBox, за исключением того, что позволяет водить информацию в маленьком поле ввода сверху ListBox. Есть несколько типов ComboBox, но наиболее популярен выпадающий вниз (drop-down combo box), который можно видеть внизу окна диалога выбора файла.

· **TScrollbar -** полоса прокрутки**,** появляется автоматически в объектах редактирования, ListBox'ах при необходимости прокрутки текста для просмотра.

· **TGroupBox** используется для визуальных целей и для указания Windows, каков порядок перемещения по компонентам на форме (при нажатии клавиши TAB).

· **TPanel** - управляющий элемент, похожий на TGroupBox, используется в декоративных целях. Чтобы использовать TPanel, просто поместите его на форму и затем положите другие компоненты на него. Теперь при перемещении TPanel будут передвигаться и эти компоненты. TPanel используется также для создания линейки инструментов и окна статуса.

TScrollBox представляет место на форме, которое можно скроллировать в вертикальном и горизонтальном направлениях. Пока Вы в явном виде не отключите эту возможность, форма сама по себе действует так же. Однако, могут быть случаи, когда понадобится прокручивать только часть формы. В таких случаях используется TScrollBox.

Это полный список объектов на первой странице Палитры Компонент. Если Вам нужна дополнительная информация, то выберите на Палитре объект и нажмите клавишу F1 - появится Справочник с полным описанием данного объекта.

Инспектор Объектов

Основное для понимания Инспектора Объектов состоит в том, что он используется для изменения характеристик любого объекта, положенного на форму. Кроме того, и для изменения свойств самой формы.
Выбрав два или более объектов одновременно, Вы можете выполнить большое число операций над ними. Например, передвигать по форме, можно выбрать пункт меню Edit | Size и установить оба поля Ширину(Width) и Высоту(Height) в Grow to Largest. Теперь оба объекта стали одинакового размера. Затем можно выбрать пункт меню Edit | Align и поставить в выравнивании по горизонтали значение Center.

Поскольку выбраны сразу два компонента, то содержимое Инспектора Объектов изменится - он будет показывать только те поля, которые являются общими для объектов. Это означает то, что изменения в свойствах, произведенные Вами, повлияют не на один, а на все выбранные объекты.
Рассмотрим изменение свойств объектов на примере свойства Color. Есть три способа изменить его значение в Инспекторе Объектов. Первый - просто напечатать имя цвета (clRed) или номер цвета. Второй путь - нажать на маленькую стрелку справа и выбрать цвет из списка. Третий путь - дважды щелкнуть на поле ввода свойства Color. При этом появится диалог выбора цвета.
Свойство Font работает на манер свойства Color. Чтобы это посмотреть, сначала выберите свойство Font для объекта TMemo и дважды щелкните мышкой на поле ввода. Появится диалог настройки шрифта, как показано на рис.12. Выберите, например, шрифт New Times Roman и установите какой-нибудь очень большой размер, например 72. Затем измените цвет фонта с помощью ComboBox'а в нижнем правом углу окна диалога. Когда Вы нажмете кнопку OK, Вы увидите, что вид текста в объекте TMemo изменился.

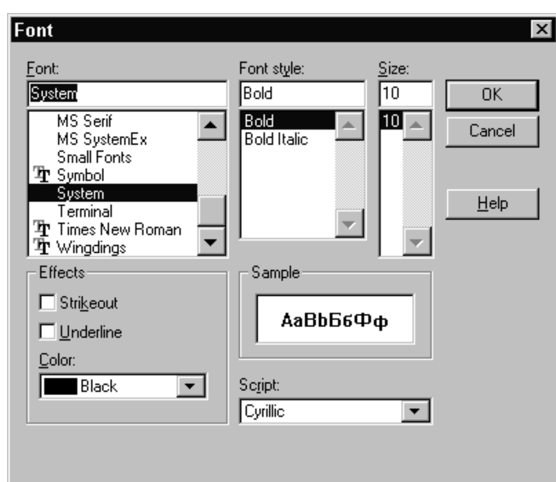


Рис.12: Диалог выбора шрифта позволяет задать тип шрифта, размер, и цвет.

В завершение краткого экскурса по Инспектору Объектов дважды щелкните на свойство Items объекта ListBox. Появится диалог, в котором Вы можете ввести строки для отображения в ListBox. Напечатайте несколько слов, по одному на каждой строке, и нажмите кнопку OK. Текст отобразится в ListBox'е.

Сохранение программы

Первый шаг - создать поддиректорию для программы. Лучше всего создать директорию, где будут храниться все Ваши программы и в ней - создать поддиректорию для данной конкретной программы. Например, Вы можете создать директорию MYCODE и внутри нее - вторую директорию TIPS1, которая содержала бы программу, над которой Вы только что работали.

После создания поддиректории для хранения Вашей программы нужно выбрать пункт меню File | Save Project. Сохранить нужно будет два файла. Первый - модуль (unit), над которым Вы работали, второй - главный файл проекта, который "владеет" Вашей программой. Сохраните модуль под именем MAIN.PAS и проект под именем TIPS1.DPR. (Любой файл с расширением PAS и словом "unit" в начале является *модулем*.)

TButton, исходный текст, заголовки и *Z*-упорядочивание

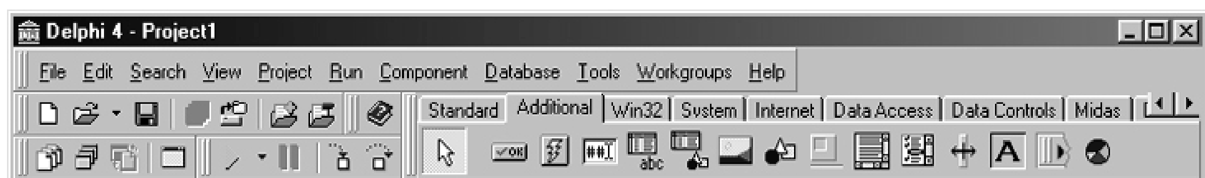
Еще несколько возможностей Инспектора Объектов и Дизайнера Форм.

Например, пусть на форму помещен объект TMemo, а затем TEdit так, чтобы он наполовину перекрывал TMemo. Теперь выберите пункт меню Edit | Send to Back, что приведет к перемещению TEdit вглубь формы, за объект TMemo. Это называется изменением *Z*-порядка компонент. Буква *Z* используется потому, что обычно математики обозначают третье измерение буквой *Z*. Так, *X* и *Y* используются для обозначения ширины и высоты, и *Z* используется для обозначения глубины.

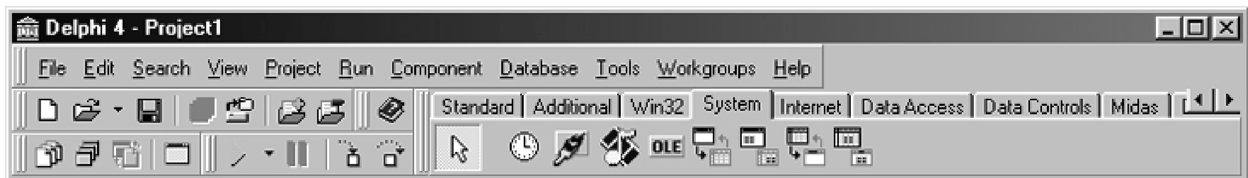
Если Вы "потеряли" на форме какой-то объект, то найти его можно в списке Combobox'а, который находится в верхней части Инспектора Объектов.

Рассмотрим также остальные страницы компонент.

Additional. Эта страница содержит более развитые компоненты. Например, компонент Outline удобен для отображения информации с иерархической структурой, а MediaPlayer позволит программам воспроизводить звук, музыку и видео. Данная страница также содержит компоненты, главное назначение которых — отображение графической информации. Компонент Image загружает и отображает растровые изображения, а компонент Shape, украсит формы окружностями, квадратами и т.д.



System. Поскольку не каждая потребность, связанная с обработкой файлов, может быть удовлетворена с помощью стандартных диалоговых окон, страница System предоставляет возможность комбинировать отдельные элементы, такие как списки дисков, каталогов и файлов. Страница System также содержит компоненты, обрабатывающие обмен высокого уровня между программами посредством OLE (Object Linking and Embedding). А компонент Timer может генерировать события через определенные, заранее установленные промежутки времени.



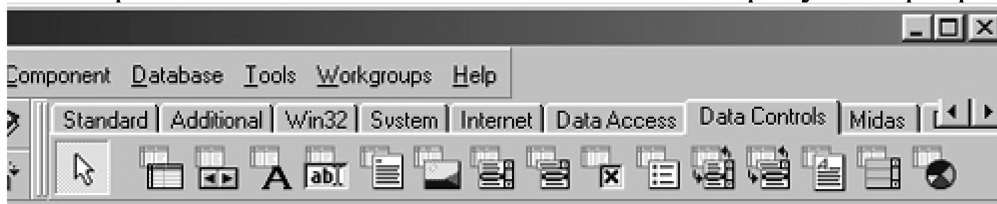
Win32. Эта страница содержит компоненты, позволяющие созданным с помощью Delphi программам использовать такие нововведения в пользовательском интерфейсе 32-разрядной Windows, как просмотр древовидных структур, просмотр списков, панель состояния, присутствующая в интерфейсе программы Windows Explorer (Проводник), расширенный текстовый редактор и др.

Dialogs. Windows 3.1 ввела в употребление стандартные диалоговые окна для операций над файлами, выбора шрифтов, цветов и т.д. Однако для использования их в обычной программе Windows может потребоваться написать немало вспомогательного кода. Страница , Dialogs предоставляет программам Delphi простой доступ к этим стандартным диалоговым окнам.

Data Access и Data Controls. Delphi использует механизм баз данных компании Borland (Borland Database Engine, BDE) для организации доступа к файлам баз данных различных форматов. Компоненты этих двух страниц облегчают программам Delphi использование сервиса баз данных, предос-

тавляемого BDE, например многопользовательского считывания, записи, индексации и выдачи запросов для таблиц dBASE и Paradox.

С использованием этих компонентов создание программы просмотра и редактирования базы данных почти не требует программирования.



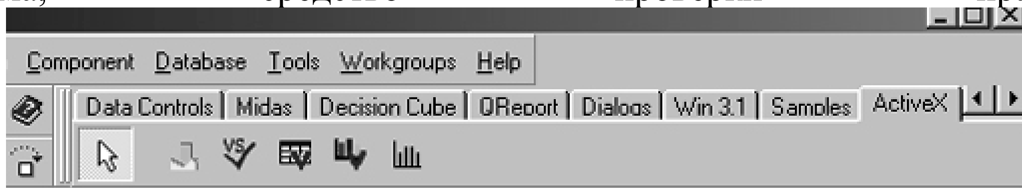
Win 3.1. На этой странице находятся компоненты Delphi 1.0, возможности которых перекрываются аналогичными компонентами Windows 95.

Internet. Эта страница предоставляет компоненты для разработки приложений, позволяющих создавать HTML-файлы непосредственно из файлов баз данных и других типов, взаимодействующих с другими приложениями для Internet. Delphi 4 дает возможность создавать приложения для Web-сервера в виде DLL-файлов : (Dynamic Link Library — Динамически компонуемая библиотека), способных содержать невизуальные компоненты. С помощью компонентов страницы Internet довольно просто создавать обработчики событий для обращения к определенному URL (Uniform Resource Locator — Унифицированный локатор ресурса), представлению документов в HTML-формате и пересылки их клиент-программе.

Samples. Эта страница содержит компоненты, которые не встроены в Delphi, но демонстрируют мощь системы компонентов. Для этих компонентов нет встроенной интерактивной справки.Все же они не менее полез-

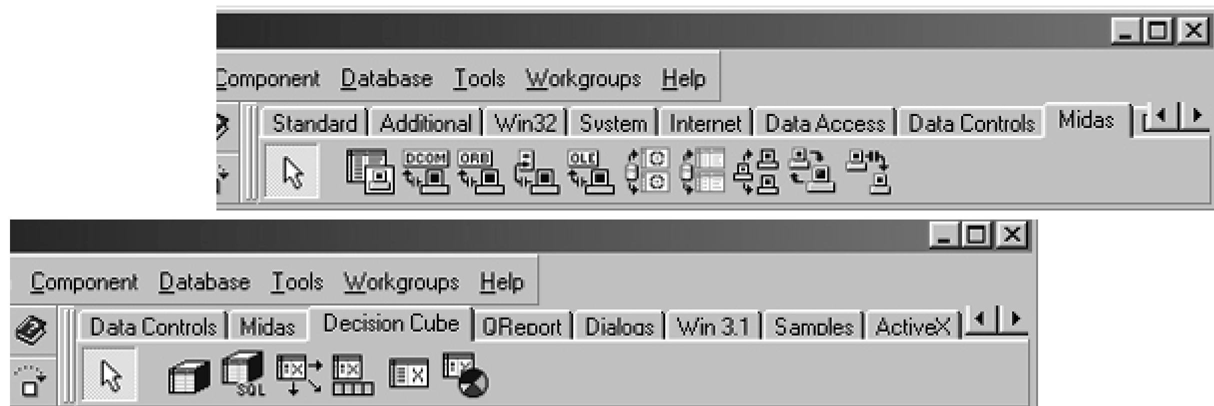
ны, чем компоненты с других страниц.

ActiveX. Эта страница содержит компоненты ActiveX, разработанные независимыми производителями программного обеспечения: сетка, диаграмма, средство проверки правописания.



QReport. Эта страница предоставляет компоненты баз данных. Здесь содержатся особые версии надписей, полей, примечаний и других элементов управления.

Midas и Decision Cube. Здесь собраны компоненты для доступа к удаленным серверам и осуществления SQL – запросов.



Рассмотрим все страницы компонент подробнее.

Страница STANDARD

На странице Standard палитры компонентов сосредоточены стандартные для Windows интерфейсные элементы, без которых не обходится практически ни одна программа.

Frame - рамка. Наравне с формой служит контейнером для размещения других компонентов. В отличие от формы может размещаться в палитре

компонентов, создавая заготовки компонентов. Впервые введен в версию Delphi 5.
MainMenu - главное меню программы. Компонент способен создавать и обслуживать сложные иерархические меню.
PopupMenu - вспомогательное или локальное меню. Обычно это меню появляется в отдельном окне после нажатия правой кнопки мыши.

Label - метка. Этот компонент используется для размещения в окне не очень длинных однострочных надписей.
Edit - строка ввода. Предназначена для ввода, отображения или редактирования одной текстовой строки. Memo - многострочный текстовый редактор. Используется для ввода и/или отображения многострочного текста.
Button - командная кнопка. Обработчик события OnClick этого компонента обычно используется для реализации некоторой команды.
CheckBox - независимый переключатель. Щелчок мышью на этом компоненте в работающей программе изменяет его логическое свойство Checked.
RadioButton - зависимый переключатель. Обычно объединяется как минимум еще с одним таким же компонентом в группу. Щелчок по переключателю приводит к автоматическому освобождению ранее выбранного переключателя в той же группе.
ListBox - список выбора. Содержит список предлагаемых вариантов (опций) и дает возможность проконтролировать текущий выбор.
ComboBox - комбинированный список выбора. Представляет собой комбинацию списка выбора и текстового редактора.
ScrollBar - полоса управления. Представляет собой вертикальную или горизонтальную полосу, напоминающую полосы прокрутки по бокам Windows-окна.
GroupBox - группа элементов. Этот компонент используется для группировки нескольких связанных по смыслу компонентов.
RadioGroup - группа зависимых переключателей. Содержит специальные свойства для обслуживания нескольких связанных зависимых переключателей.
Panel - панель. Этот компонент, как и GroupBox, служит для объединения нескольких компонентов. Содержит внутреннюю и внешнюю кромки, что позволяет создать эффекты “вдавленности” и “выпуклости”.
Actiontist - список действий. Служит для централизованной реакции программы на действия пользователя, связанные с выбором одного из группы однотипных управляющих элементов таких как опции меню, пиктографические кнопки и т. п. Впервые, введен в версии Delphi 4.
Страница ADDITIONAL
В страницу Additonal помещены 18 дополнительных компонентов, с помощью которых можно разнообразить вид диалоговых окон.

BitBtn - командная кнопка с надписью и пиктограммой.
SpeedButton - пиктографическая кнопка. Обычно используется для быстрого доступа к тем или иным опциям главного меню.
MaskEdit - специальный текстовый редактор. Способен фильтровать вводимый текст, например, для правильного ввода даты.
StringGrid - таблица строк. Этот компонент обладает мощными возможностями для представления текстовой информации в табличном виде.
DrawGrid - произвольная таблица. В отличие от StringGrid ячейки этого компонента могут содержать произвольную информацию, в том числе и рисунки.
Image - рисунок. Этот компонент предназначен для отображения рисунков, в том числе пиктограмм и метафайлов.
Shape - фигура. С помощью этого компонента вы можете вставить в окно правильную геометрическую фигуру - прямоугольник, эллипс, окружность и т. п.
Bevel - кромка. Служит для выделения отдельных частей окна трехмерными рамками или полосами.
ScrollBox - панель с полосами прокрутки. В отличие от компонента Panel автоматически вставляет полосы прокрутки, если размещенные в нем компоненты отсекаются его границами.
CheckListBox - список множественного выбора. Отличается от стандартного компонента ListBox наличием рядом с каждой опцией независимого переключателя типа CheckBox, облегчающего выбор сразу нескольких опций. Впервые введен в версии 3.
Splitter - граница. Этот компонент размещается на форме между двумя другими видимыми компонентами и дает возможность пользователю во время прогона программы перемещать границу, отделяющую компоненты друг от друга. Впервые введен в версии 3.
StaticText - статический текст. Отличается от стандартного компонента Label наличием собственного windows-окна, что позволяет обводить текст рамкой или выделять его в виде “вдавленной” части формы. Впервые введен в версии 3.
ControiBar - полоса управления. Служит контейнером для “причаливаемых” компонентов в технологии Drag&Dock. Впервые введен в версии 4.
ApplicationEvents - получатель события. Если этот компонент помещен на форму, он будет получать все предназначенные для программы сообщения Windows (без этого компонента сообщения принимает глобальный объект-программа Application). Впервые введен в версии 5.
ValueListEditor - редактор строк, содержащих пары имя = значение. Пары такого типа широко используются в Windows, например, в файлах инициации, в системном реестре и т. п. Впервые введен в версии 6.

LabeledEdit - комбинация однострочного редактора и метки. Впервые введен в версии 6.
ColorBox - специальный вариант ComboBox для выбора одного из системных цветов. Впервые введен в версии 6.
Chart - диаграмма. Этот компонент облегчает создание специальных панелей для графического представления данных. Впервые введен в версии 3.
ActionManager - менеджер действий. Совместно с тремя следующими компонентами обеспечивает создание приложений, интерфейс которых (главное меню и инструментальные кнопки) может настраиваться пользователем. Впервые введен в версии 6.
ActionMainMenuBar - полоса меню, опции которого создаются с помощью компонента ActionManager. Впервые введен в версии 6.
ActionToolBar - полоса для размещения пиктографических кнопок, создаваемых с помощью компонента ActionManager. Впервые введен в версии 6.
CustomizeDig - диалог настройки. С помощью этого компонента пользователь может по своему вкусу настроить интерфейс работающей программы. Впервые введен в версии 6.

Страница WIN32

Страница Win32 содержит интерфейсные элементы для 32-разрядных операционных систем Windows 95/98/NT/2000 (в версии 2 эта страница называется win 95). Этой страницы нет в версии 1.
TabControl - набор закладок. Каждая закладка представляет собой прямоугольное поле с надписью и/или рисунком. Выбор той или иной закладки распознается программой и используется для управления содержимым окна компонента.
PageControl - набор панелей с закладками. Каждая панель может содержать свой набор интерфейсных элементов и выбирается щелчком по связанной с ней закладке.
ImageList - набор рисунков. Представляет собой хранилище для нескольких рисунков одинакового размера.
RichEdit - многострочный редактор форматированного текста. В отличие от компонента Memo страницы Standard текст в компоненте RichEdit подчиняется правилам Расширенного Текстового Формата (RTF - Rich Text Format) и может изменять такие свои характеристики, как шрифт, цвет, выравнивание и т. д.
TrackBar - регулятор. Используется для управления значениями некоторых величин в программах. Например, с его помощью удобно изменять громкость звучания в мультимедийных программах.
ProgressBar - индикатор процесса. С помощью этого компонента можно отображать ход исполнения достаточно длительного по времени процесса, например, процесса переноса данных на дискету.

UpDown - цифровой регулятор. Две кнопки этого компонента предназначены для увеличения (верхняя) или уменьшения (нижняя) связанной с компонентом числовой величины.
HotKey - управляющая клавиша. Компонент используется для ввода управляющих клавиш, таких как F1, Alt+A, Ctrl+Shift+1 и т. п.
Animate - мультипликатор. Предназначен для отображения последовательно сменяющих друг друга кадров движущихся изображений (видеоклипов). Компонент не может сопровождать видеоклип звуком. Впервые введен в версии 3.
DateTimePicker - селектор времени/даты. Этот компонент предназначен для ввода и отображения даты или времени. Впервые введен в версии 3.
TreeView - дерево выбора. Представляет собой совокупность связанных в древовидную структуру пиктограмм. Обычно используется для просмотра структуры каталогов (папок) и других подобных элементов, связанных иерархическими отношениями.
ListView - панель пиктограмм. Организует просмотр нескольких пиктограмм и выбор нужной. Этот компонент способен располагать пиктограммы в горизонтальных или вертикальных рядах и показывать их в крупном или мелком масштабе.
HeaderControl- управляющий заголовок. Представляет собой горизонтальную или вертикальную полосу, разделенную на ряд смежных секций с надписями. Размеры секций можно менять мышью на этапе работы программы. Обычно используется для изменения размеров столбцов или строк в разного рода таблицах.
StatusBar - панель статуса. Предназначена для размещения разного рода служебной информации в окнах редактирования. Посмотрите на нижнюю часть рамки окна кода Delphi или текстового редактора Word, и вы увидите этот компонент в действии.
ToolBar - инструментальная панель. Этот компонент служит контейнером для командных кнопок BitBtn и способен автоматически изменять их размеры и положение при удалении кнопок или при добавлении новых. Впервые введен в версии 3.
CoolBar - инструментальная панель. В отличие от ToolBar используется как контейнер для размещения стандартных интерфейсных компонентов Windows, таких как Edit, ListBox, ComdoBox и т. д. Впервые введен в версии 3.
PageScroller - прокручиваемая панель. Служит для размещения узких инструментальных панелей. При необходимости автоматически создает по краям панели стрелки прокрутки. Впервые введен в версии 4.
ComboBoxEx - компонент в функциональном отношении подобен comboBox (страница standard), но может отображать в выпадающем списке небольшие изображения. Впервые введен в версии 6.

Страница SYSTEM

На этой странице представлены компоненты, которые имеют различное функциональное назначение, в том числе компоненты, поддерживающие стандартные для Windows технологии межпрограммного обмена данными OLE (Object Linking and Embedding -связывание и внедрение объектов) и DDE (Dynamic Data Exchange -динамический обмен данными).

Timer - таймер. Этот компонент служит для отсчета интервалов реального времени.

PaintBox - окно для рисования. Создает прямоугольную область, предназначенную для прорисовки графических изображений.

MediaPlayer - мультимедийный проигрыватель. С помощью этого компонента можно управлять различными мультимедийными устройствами.

OleContainer - OLE-контейнер. Служит приемником связываемых или внедряемых объектов.

Компоненты этой страницы имеются во всех предыдущих версиях Delphi.

Страница DIALOGS

Компоненты страницы Dialogs реализуют стандартные для Windows диалоговые окна.

OpenDialog - открыть. Реализует стандартное диалоговое окно “Открыть файл”.

SaveDialog - сохранить. Реализует стандартное диалоговое окно “Сохранить файл”.

OpenPictureDialog - открыть рисунок. Реализует специальное окно выбора графических файлов с возможностью предварительного просмотра рисунков.

SavePictureDialog - сохранить рисунок. Реализует специальное окно сохранения графических файлов с возможностью предварительного просмотра рисунков.

FontDialog - шрифт. Реализует стандартное диалоговое окно выбора шрифта.

ColorDialog - цвет. Реализует стандартное диалоговое окно выбора цвета.

PrintDialog - печать. Реализует стандартное диалоговое окно выбора параметров для печати документа.

PrinterSetupDialog - настройка принтера. Реализует стандартное диалоговое окно для настройки печатающего устройства.

FindDialog - поиск. Реализует стандартное диалоговое окно поиска текстового фрагмента.

ReplaceDialog - замена. Реализует стандартное диалоговое окно поиска и замены текстового фрагмента.

Компоненты OpenPictureDialog И SavePictureDialog введены в версии 3, остальные имеются в предыдущих версиях. Разумеется, интерфейс окон для Windows 16 (версия 1) отличается от интерфейса Windows 32.

Страница WIN31

Большинство компонентов этой страницы введены для совместимости с версией 1. В современных программах вместо них рекомендуется использовать соответствующие компоненты страницы Win32.

TabSet - набор закладок. В приложениях для Windows 32 вместо него рекомендуется использовать компонент TabControl.

OutLine - дерево выбора. В приложениях для Windows 32 вместо него рекомендуется использовать компонент Treeview.

TabbedNotebook - набор панелей с закладками. В приложениях для Windows 32 вместо него рекомендуется использовать компонент PageControl.

Notebook - набор панелей без закладок. В приложениях для Windows 32 вместо него рекомендуется использовать компонент Page-Control.

Header - управляющий заголовок. В приложениях для Windows 32 вместо него рекомендуется использовать компонент Header-Control.

FileListBox - панель выбора файлов.

DirectorybistBox - панель выбора каталогов.

DiriveComboBox - панель выбора дисков.

FilterComboBox - панель фильтрации файлов.

Компоненты FileListBox, DirectoryListBox, DriveComboBox и FilterComboBox впервые появились в версии 3. Их функции реализованы элементами стандартных окон OpenDialog и SaveDialog, которые и рекомендуется использовать в Windows 32.

Страница SAMPLES

Эта страница содержит компоненты разного назначения.

Gauge - индикатор состояния. Подобен компоненту ProgressBar (страница Win32), но отличается большим разнообразием форм.

ColorGrid - таблица цветов. Этот компонент предназначен для выбора основного и фонового цветов из 16-цветной палитры.

SpinButton - двойная кнопка. Дает удобное средство управления некоторой числовой величиной.

SpinEdit - редактор числа. Обеспечивает отображение и редактирование целого числа с возможностью его изменения с помощью двойной кнопки.

DirectoryOutLine - список каталогов. Отображает в иерархическом виде структуру каталогов дискового накопителя.

Calendar - календарь. Предназначен для показа и выбора дня в месяце.

Страница ACTIVE X

Компоненты ActiveX являются “чужими” для Delphi: они создаются другими инструментальными средствами разработки программ (например, C++ или Visual Basic) и внедряются в Delphi с помощью технологии OLE. На странице ActiveX представлены лишь 4 из великого множества ActiveX-компонентов, разрабатываемых повсюду в мире компаниями -

производителями программных средств и отдельными программистами. Chartfx - интерактивный график. Дает программисту удобное средство включения в программу интерактивных (диалоговых) графиков.

VSSpell - спеллер. Осуществляет орфографическую проверку правильности написания английских слов.

F1Book - электронная таблица. Позволяет создавать и использовать рабочие книги электронных таблиц, подобно тому как это делает MS Excel.

VtChart - мастер диаграмм. Обеспечивает мощные средства построения двух- и трехмерных диаграмм по результатам табличных вычислений.

Компоненты для работы с базами данных

В Delphi развиты средства построения приложений, рассчитанных на работу с электронными архивами (базами данных). Причем Delphi 6 предоставляет программисту возможность выбора способа доступа к данным: это может быть стандартный для ранних версий Delphi доступ с помощью машины баз данных BDE (Borland Data base Engine), усиленно развиваемая Microsoft технология ADO ActiveX Data Objects), прямое управление сервером InterBase с помощью технологии IBExpress, наконец, технология dbExpress для непосредственного обращения к промышленным серверам MySQL, DB2, Oracle и некоторым другим.

Страница Data Access

В отличие от предыдущих версий на этой странице собраны компоненты, которые не зависят от используемого доступа к базе данных (большинство компонентов с этой страницы предыдущих версий перекочевали на страницу bde). Они в основном используются в так называемых трехзвенных БД (с сервером приложений). Часть компонентов известны по другим версиям Delphi, часть появилась в версии 6.

Страница Data Controls

15 компонентов этой страницы предназначены для визуализации данных, их ввода и редактирования. Многие компоненты этой страницы введены еще в версии 1.

Страница dbExpress

7 компонентов, представленных на этой странице, поддерживают технологию dbExpress прямого доступа к некоторым промышленным серверам баз данных. Все компоненты страницы впервые введены в версии 6.

Страница DataSnap

На этой странице сосредоточены компоненты, реализующие взаимодействие машин в локальной сети или Интернет в типичном для БД случае, когда клиент работает с удаленными данными. Часть компонентов известны по другим версиям Delphi, часть появилась в версии 6.

Страница BDE

Здесь представлены компоненты, поддерживающие доступ к данным с помощью BDE - Table, Query, StoredProc И Т. П. Механизм BDE в равной

степени успешно работает как с файл-серверными, кис клиент-серверными БД. Компоненты этой страницы есть во всех версиях Delphi.
Страница ADO
Компоненты этой страницы в функциональном отношении во многом подобны компонентам страницы BDE, но поддерживают доступ к данным с помощью техологии ADO (ADOTable, ADOQuery, ADostoredproc и т. д.). Все компоненты страницы впервые введены в версии 5.
Страница InterBase
"Родной" для Delphi сервер баз данных InterBase (производитель - InterBase Software Corporation - является дочерним предприятием Borland) имеет непосредственную поддержку в виде компонентов этой страницы. В них используется технология IBExpress, позволяющая отказаться от BDE, ADO или иных подобных механизмов доступа к данным. Все компоненты страницы впервые введены в версии 5.
Страница Decision Cube
На этой странице представлены компоненты для систем принятия решений на основании анализа многомерных наборов данных. Компоненты этой страницы впервые введены в версии 3.
Страница QReport
Около 30 компонентов страницы предназначены для упрощения создания отчетов по материалам, хранящимся в БД. Большинство компонентов страницы впервые введено в версии 2.
Компоненты для доступа к интернет
Страница Internet
Компоненты этой страницы обеспечивают средства связи программы с глобальной компьютерной сетью Интернет. Эта сеть позволяет установить соединение между двумя удаленными компьютерами, один из которых (клиент) получает информацию, а другой (сервер) передает ее. Оба компьютера должны следовать протоколу TCP/IP (Transport Control Protocol/Internet Protocol - транспортный управляющий протокол/Интернет протокол), обеспечивающему логическую независимость связи от аппаратных средств компьютеров. Частью Интернет является всемирная паутина World Wide Web (WWW), использующая межкомпьютерный обмен так называемыми HTML-страницами (HyperText Markup Language - язык разметки гипертекста) на основе HTTP-протокола (HyperText Transfer Protocol - протокол передачи гипертекста). WWW реализует удобные средства для неформального общения мирового сообщества на самые разные темы. В то же время возможности Интернет не ограниваются только WWW, т. к. по глобальной сети можно передавать гктронную почту, разнообразные файлы, устраивать телеконференции и даже осуществлять телефонные переговоры. В последнее время усиленно развиваются так называемые интранет-сети, в ко-торых технология Интернет используется для

передачи служебной и ювой информации в рамках одного или нескольких предприятий. Для создания именно таких сетей и предназначены в основном компоненты этой страницы.

Страница FastNet

Компоненты этой страницы предоставляют программисту возможность использования различных протоколов для передачи деловых сообщений и данных по локальным и/или глобальным сетям, в том числе и по Интернет. В версиях 2, 3 и 4 они размещались на странице internet. В версии 1 таких компонентов нет. Для межплатформенных программ вместо этих компонентов следует использовать компоненты страниц Indy.

Страница WebServices

Компоненты этой страницы поддерживают технологию SOAP Simple Object Access Protocol для создания служб Web. Служба Web - это программа, запускаемая сервером Web в ответ на клиентское требование. Служба должна подготовить отклик, который она возвращает серверу, а тот - клиенту.

Страница WebSnap

На этой странице сосредоточены компоненты, развивающую известную из предыдущих версий технологию Web Server. В настоящей версии эти компоненты не поддерживают межплатформенные программы.

Страницы Indy Clients, Indy Servers, Indy Misc

Расположенные на этих страницах компоненты в функциональном плане дублируют компоненты страницы FastNet, но позволяют их использовать в межплатформенных приложениях.

Доступ к серверам автоматизации

Многочисленные компоненты страницы servers обеспечивают удобный программный доступ к популярным COM-серверам, входящим в Microsoft Office'97 и доступным на любом компьютере, на котором полностью или частично установлен комплект этих программ.

Поскольку в базовом языке MS Qffice'97 - Visual Basic for Application - произошли значительные изменения по сравнению с версией, использовавшейся в MS Office'95 компоненты этой страницы будут нормально работать только с MS Office'97

Использование этих компонентов осложняется двумя обстоятельствами. Во-первых, на вашей машине или в доступном вам сетевом окружении должны быть установлены соответствующие серверы (Word, Excel, PowerPoint и т. д.). Во-вторых, все эти компоненты представляют собой так называемые контроллеры Автоматизации, т. е. существеннейшим образом используют многочисленные свойства, методы и события своих серверов. Поскольку взаимодействие с такого рода серверами требует знания интерфейса сервера, каждый компонент, с одной стороны, одинаков, показывая в окне Инспектора объектов лишь минимум свойств, общих для всех сер-

веров Автоматизации, а с другой - разительно отличается от остальных своим уникальным набором свойств, методов и событий.

2.2 Свойства в Delphi

Каждый компонент, помещенный на форму, имеет свое отражение в окне Инспектора Объектов (Object Inspector). Object Inspector имеет две "странички" - "Properties" (Свойства) и "Events" (События). Создание программы в Delphi сводится к "нанесению" компонент на форму (которая, кстати, также является компонентом) и настройке взаимодействия между ними путем:
- изменения значения свойств этих компонент
- написания адекватных реакций на события.

Свойство является важным атрибутом компонента. Для программиста свойство выглядит как простое поле какой-либо структуры, содержащее некоторое значение. Однако, в отличие от "просто" поля, любое изменение значения некоторого свойства любого компонента сразу же приводит к изменению визуального представления этого компонента, поскольку свойство инкапсулирует в себе методы (действия), связанные с чтением и записью этого поля (которые, в свою очередь, включают в себя необходимую перерисовку). Свойства служат двум главным целям. Во-первых, они определяют внешний вид формы или компонента. А во-вторых, свойства определяют поведение формы или компонента.

Существует несколько типов свойств, в зависимости от их "природы", т.е. внутреннего устройства.

1. Простые свойства - это те, значения которых являются числами или строками. Например, свойства Left и Top принимают целые значения, определяющие положение левого верхнего угла компонента или формы. Свойства Caption и Name представляют собой строки и определяют заголовок и имя компонента или формы.
2. Перечислимые свойства - это те, которые могут принимать значения из предопределенного набора (списка). Простейший пример - это свойство типа Boolean, которое может принимать значения True или False.
3. Вложенные свойства - это те, которые поддерживают вложенные значения (или объекты). Object Inspector изображает знак "+" слева от названия таких свойств. Имеется два вида таких свойств: множества и комбинированные значения. Object Inspector изображает множества в квадратных скобках. Если множество пусто, оно отображается как []. Установки для вложенных свойств вида "множество" обычно имеют значения типа Boolean. Наиболее распространенным примером такого свойства является свойство Style с вложенным множеством булевых значений. Комбинированные значения отображаются в Инспекторе Объектов как коллекция не-

которых величин, каждый со своим типом данных. Некоторые свойства, например, Font, для изменения своих значений имеют возможность вызвать диалоговое окно. Для этого достаточно щелкнуть маленькую кнопку с тремя точками в правой части строки Инспектора Объектов, показывающей данное свойство.
С другой стороны, в режиме выполнения программист имеет возможность не только манипулировать всеми свойствами, отображаемыми в Инспекторе Объектов, но и управлять более обширным их списком.
Управление свойствами визуальных компонент в режиме выполнения.
Все изменения значений свойств компонент в режиме выполнения должны осуществляться путем прямой записи строк кода на языке Паскаль. В режиме выполнения невозможно использовать Object Inspector. Однако, доступ к свойствам компонентов довольно легко получить программным путем. Все, что Вы должны сделать для изменения какого-либо свойства - это написать строчку кода.
Объектно-ориентированный язык Паскаль, лежащий в основе Delphi, в качестве базового имеет принцип соответствия визуальных компонент тем вещам, которые они представляют. Разработчики Delphi поставили перед собой цель, чтобы, например, представление компонента Button (кнопка), инкапсулирующее некий код, соответствовало визуальному изображению кнопки на экране и являлось как можно более близким эквивалентом реальной кнопки, которую Вы можете найти на клавиатуре. И именно из этого принципа родилось понятие свойства.

2.3 Методы в Delphi

Класс - это категория объектов, обладающих одинаковыми свойствами и поведением. При этом объект представляет собой просто экземпляр какого-либо класса. Например, в Delphi тип “форма” (окно) является классом, а переменная этого типа - объектом. Метод - это процедура, которая определена как часть класса и инкапсулирована (содержится) в нем. Методы манипулируют полями и свойствами классов (хотя могут работать и с любыми переменными) и имеют автоматический доступ к любым полям и методам своего класса. Доступ к полям и методам других классов зависит от уровня “защищенности” этих полей и методов. Методы можно создавать как визуальными средствами, так и путем написания кода вручную.
Каждая программа в Delphi состоит из файла проекта, имеющего расширение .DPR и одного или нескольких модулей, имеющих расширение .PAS. Модуль, в котором содержится главная форма проекта, называется головным. Указанием компилятору о связях между модулями является предложение Uses, которое определяет зависимость модулей.

Нет никакого функционального различия между модулями, созданными в Редакторе, и модулями, сгенерированными Delphi автоматически. В любом случае модуль подразделяется на три секции:
· Заголовок
· Секция Interface
· Секция Implementation
Таким образом, “скелет” модуля выглядит следующим образом:
unit Main; {Заголовок модуля}
interface {Секция Interface}
implementation {Секция Implementation}
end.
В интерфейсной секции (interface) описывается все то, что должно быть видимо для других модулей (типы, переменные, классы, константы, процедуры, функции). В секции implementation помещается код, реализующий классы, процедуры или функции.
В Delphi процедурам и функциям (а, следовательно, и методам классов) могут передаваться параметры для того, чтобы обеспечить их необходимой для работы информацией.
Заголовок процедуры состоит из пяти частей:
· Первая часть - зарезервированное слово “procedure”; пятая часть - концевая точка с запятой “;”. Обе эти части служат определенным синтаксическим целям, а именно: первая информирует компилятор о том, что определен синтаксический блок “процедура”, а вторая указывает на окончание заголовка (собственно говоря, все операторы в Delphi должны заканчиваться точкой с запятой).
· Вторая часть заголовка - слово “TForm1”, которое квалифицирует то обстоятельство, что данная процедура является методом класса TForm1.
· Третья часть заголовка - имя процедуры. Вы можете выбрать его любым, по вашему усмотрению.
· Четвертая часть заголовка - параметр. Параметр декларируется внутри скобок и, в свою очередь, состоит из двух частей. Первая часть - имя параметра, вторая - его тип. Эти части разделены двоеточием. Если Вы описываете в процедуре более чем один параметр, нужно разделить их точкой с запятой, например:
procedure TForm1.WriteAll(NewString: String);
procedure Example(Param1: String; Param2: String);
После того как создали “вручную” заголовок процедуры, являющейся методом класса, нужно включить его в декларацию класса, например, путем копирования (для методов, являющихся откликами на дельфийские события, данное включение производится автоматически).
<u>Информация периода выполнения.</u>

Открытость среды Delphi позволяет получать и оперировать информацией особого рода, называемой информацией периода выполнения (RTTI - run-time type information). Эта информация организована в виде нескольких уровней.

1. Верхний уровень RTTI представлен как средство проверки и приведения типов с использованием ключевых слов is и as.

Ключевое слово is дает программисту возможность определить, имеет ли данный объект требуемый тип или является одним из наследников данного типа, например, таким образом:

```
if MyObject is TSomeObj then ...
```

Имеется возможность использовать RTTI и для процесса приведения объектного типа, используя ключевое слово as:

```
if MyObject is TSomeObj then
(MyObject as TSomeObj).MyField:=...
```

что эквивалентно:

```
TSomeObj(MyObject).MyField:=...
```

2. Средний уровень RTTI использует методы объектов и классов для подмены операций as и is на этапе компиляции. В основном, все эти методы заложены в базовом классе TObject, от которого наследуются все классы библиотеки компонент VCL. Для любого потомка TObject доступны, в числе прочих, следующие информационные методы:

- ClassName - возвращает имя класса, экземпляром которого является объект;
- ClassInfo - возвращает указатель на таблицу с RTTI, содержащей информацию о типе объекта, типе его родителя, а также о всех его публикуемых свойствах, методах и событиях;
- ClassParent - возвращает тип родителя объекта;
- ClassType - возвращает тип самого объекта;
- InheritsFrom - возвращает логическое значение, определяющее, является ли объект потомком указанного класса;
- InstanceSize - возвращает размер объекта в байтах.

Эти методы могут использоваться в программном коде напрямую.

3. Нижний уровень RTTI определяется в дельфийском модуле TypInfo и представляет особый интерес для разработчиков компонент. Через него можно получить доступ к внутренним структурам Delphi, в том числе, к ресурсам форм, инспектору объектов и т.п.

Итак, доступ к информации периода выполнения в Delphi позволяет динамически получать как имя объекта, находящегося на форме, так и название класса, которому он принадлежит. Для этого используется свойство Name, имеющееся у любого класса-наследника TComponent (а таковыми являются все компоненты, входящие в дельфийскую библиотеку VCL), и метод ClassName, доступный для любого потомка класса базового TObject. А, по-

скольку класс TComponent, в свою очередь, является наследником класса TObject, то он доступен для всех компонент из библиотеки VCL.

Вопросы для самоподготовки:

1. Что такое классы и методы объектов?
2. Какие свойства может иметь объект «форма»?
3. Какие компоненты расположены на странице Dialogs?
4. Какие компоненты расположены на странице System?
5. Что такое проект в Дельфи и какие файлы его реализуют?

Глава 3. Обработка исключительных ситуаций в Delphi

3.1 Структурная обработка исключительных ситуаций

Структурная обработка исключительных ситуаций - это система, позволяющая программисту при возникновении ошибки (исключительной ситуации) связаться с кодом программы, подготовленным для обработки такой ошибки. Это выполняется с помощью языковых конструкций, которые как бы “охраняют” фрагмент кода программы и определяют обработчики ошибок, которые будут вызываться, если что-то пойдет не так в “охраняемом” участке кода. В данном случае понятие исключительной ситуации относится к языку и не нужно его путать с системными исключительными ситуациями (hardware exceptions), такими как General Protection Fault. Эти исключительные ситуации обычно используют прерывания и особые состояния “железа” для обработки критичной системной ошибки; исключительные ситуации в Delphi же независимы от “железа”, не используют прерываний и используются для обработки ошибочных состояний, с которыми подпрограмма не готова иметь дело. Системные исключительные ситуации, конечно, могут быть перехвачены и преобразованы в языковые исключительные ситуации, но это только одно из применений языковых исключительных ситуаций.

При традиционной обработке ошибок, ошибки, обнаруженные в процедуре обычно передаются наружу (в вызывавшую процедуру) в виде возвращаемого значения функции, параметров или глобальных переменных (флажков). Каждая вызывающая процедура должна проверять результат вызова на наличие ошибки и выполнять соответствующие действия. Часто, это просто выход еще выше, в более верхнюю вызывающую процедуру и т.д. : функция А вызывает B, B вызывает C, C обнаруживает ошибку и возвращает код ошибки в B, B проверяет возвращаемый код, видит, что возникла ошибка и возвращает код ошибки в A, A проверяет возвращаемый код и выдает сообщение об ошибке либо решает сделать что-нибудь еще, раз первая попытка не удалась.

Одна ошибка в коде программы или переприсвоение в цепочке возвращаемых значений может привести к тому, что нельзя будет связать ошибочное состояние с положением дел во внешнем мире. Результатом будет ненормальное поведение программы, потеря данных или ресурсов, или крах системы.

Структурная обработка исключительной ситуации замещает ручную обработку ошибок автоматической, сгенерированной компилятором системой уведомления. В приведенном выше примере, процедура A установила бы "охрану" со связанным обработчиком ошибки на фрагмент кода, в котором вызывается B. B просто вызывает C. Когда C обнаруживает ошибку, то создает (raise) исключительную ситуацию. Специальный код, сгенерированный компилятором и встроенный в Run-Time Library (RTL) начинает поиск обработчика данной исключительной ситуации. При поиске "защищенного" участка кода используется информация, сохраненная в стеке. В процедурах C и B нет такого участка, а в A - есть. Если один из обработчиков ошибок, которые используются в A, подходит по типу для возникшей в C исключительной ситуации, то программа переходит на его выполнение. При этом область стека, используемая в B и C, очищается; выполнение этих процедур прекращается.

Если в A нет подходящего обработчика, то поиск продолжается в более верхнем уровне, и так может идти, пока поиск ни достигнет подходящего обработчика ошибок среди используемых по умолчанию обработчиков в RTL. Обработчики ошибок из RTL только показывают сообщение об ошибке и форсированно прекращают выполнение программы. Любая исключительная ситуация, которая осталась необработанной, приведет к прекращению выполнения приложения.

Без проверки возвращаемого кода после каждого вызова подпрограммы, код программы должен быть более простым, а скомпилированный код - более быстрым. При наличии исключительных ситуаций подпрограмма B не должна содержать дополнительный код для проверки возвращаемого результата и передачи его в A. B ничего не должна делать для передачи исключительной ситуации, возникшей в C, в процедуру A - встроенная система обработки исключительных ситуаций делает всю работу.

Данная система называется структурной, поскольку обработка ошибок определяется областью "защищенного" кода; такие области могут быть вложенными. Выполнение программы не может перейти на произвольный участок кода; выполнение программы может перейти только на обработчик исключительной ситуации активной программы.

Модель исключительных ситуаций в Delphi.

Модель исключительных ситуаций в Object Pascal является невозобновляемой(non-resumable). При возникновении исключительной ситуации Вы уже не сможете вернуться в точку, где она возникла, для продолжения вы-

полнения программы (это позволяет сделать возобновляемая (resumable) модель). Невозобновляемые исключительные ситуации разрушают стек, поскольку они сканируют его в поисках обработчика; в возобновляемой модели необходимо сохранять стек, состояние регистров процессора в точке возникновения ошибки и выполнять поиск обработчика и его выполнение в отдельном стеке. Возобновляемую систему обработки исключительных ситуаций гораздо труднее создать и применять, нежели невозобновляемую.

Синтаксис обработки исключительных ситуаций.

Новое ключевое слово, добавленное в язык Object Pascal - try. Оно используется для обозначения первой части защищенного участка кода. Существует два типа защищенных участков:

try..except
try..finally

Первый тип используется для обработки исключительных ситуаций. Его синтаксис:

```
try
Statement 1;
Statement 2;
...
except
on Exception1 do Statement;
on Exception2 do Statement;
...
else
Statements; {default exception-handler}
end;
```

Для уверенности в том, что ресурсы, занятые приложением, освободятся в любом случае, можно использовать конструкцию второго типа. Код, расположенный в части finally, выполняется в любом случае, даже если возникает исключительная ситуация. Соответствующий синтаксис:

```
try
Statement1;
Statement2;
...
finally
Statements; {These statements always execute}
end;
```

Декларация типа 'ESampleError =class' вместо '=object'; это новое расширение языка. Delphi вводит новую модель объектов, доступную через декларацию типа '=class'. Исключительные ситуации (exceptions) являются классами, частью новой объектной модели.

Конструкция try..except подходит, если известно, какой тип ошибок нужно обрабатывать в конкретной ситуации. Но что делать, если требуется выполнить некоторые действия в любом случае, произошла ошибка или нет? Это тот случай, когда понадобится конструкция try..finally.
Оба типа конструкции try можно использовать в любом месте, допускается вложенность любой глубины. Исключительную ситуацию можно вызывать внутри обработчика ошибки, конструкцию try можно использовать внутри обработчика исключительной ситуации.
Вызов исключительной ситуации
Например, процедура, которая вызывает исключительную ситуацию:
raise ESampleError.Create('Error!');
После ключевого слова raise следует код. В момент вызова исключительной ситуации создается экземпляр указанного класса; данный экземпляр существует до момента окончания обработки исключительной ситуации и затем автоматически уничтожается. Вся информация, которую нужно сообщить в обработчик ошибки передается в объект через его конструктор в момент создания.
Почти все существующие классы исключительных ситуаций являются наследниками базового класса Exception и не содержат новых свойств или методов. Класс Exception имеет несколько конструкторов, какой из них конкретно использовать - зависит от задачи.
Доступ к экземпляру объекта exception.
При вызове исключительной ситуации (raise) автоматически создается экземпляр соответствующего класса, который и содержит информацию об ошибке. Вопрос в том, как в обработчике данной ситуации получить доступ к этому объекту.
Еще одно новшество в языке Object Pascal - создание локальной переменной. Есть еще один способ доступа к экземпляру exception - использовать функцию ExceptionObject:
on ESampleError do
writeln(ESampleError(ExceptionObject).Message);

3.2 Предопределенные обработчики исключительных ситуаций

Exception - базовый класс-предок всех обработчиков исключительных ситуаций.
EAbort - “скрытое” исключение. Используйте его тогда, когда хотите прервать тот или иной процесс с условием, что пользователь программы не должен видеть сообщения об ошибке. Для повышения удобства использования в модуле SysUtils предусмотрена процедура Abort, определенная, как:
procedure Abort;

```
begin
raise EAbort.CreateRes(SOperationAborted) at ReturnAddr;
end;
```

EComponentError - вызывается в двух ситуациях:

1) при попытке регистрации компоненты за пределами процедуры Register;

2) когда имя компоненты не уникально или не допустимо.

EConvertError - происходит в случае возникновения ошибки при выполнении функций StrToInt и StrToFloat, когда конвертация строки в соответствующий числовой тип невозможна.

EInOutError - происходит при ошибках ввода/вывода при включенной директиве {$I+}.

EIntError - предок исключений, случающихся при выполнении целочисленных операций.

EDivByZero - вызывается в случае деления на ноль, как результат RunTime Error 200.

EIntOverflow - вызывается при попытке выполнения операций, приводящих к переполнению целых переменных, как результат RunTime Error 215 при включенной директиве {$Q+}.

ERangeError - вызывается при попытке обращения к элементам массива по индексу, выходящему за пределы массива, как результат RunTime Error 201 при включенной директиве {$R+}.

EInvalidCast - происходит при попытке приведения переменных одного класса к другому классу, несовместимому с первым (например, приведение переменной типа TListBox к TMemo).

EInvalidGraphic - вызывается при попытке передачи в LoadFromFile файла, несовместимого графического формата.

EInvalidGraphicOperation - вызывается при попытке выполнения операций, неприменимых для данного графического формата (например, Resize для TIcon).

EInvalidObject - реально нигде не используется, объявлен в Controls.pas.

EInvalidOperation - вызывается при попытке отображения или обращения по Windows-обработчику (handle) контрольного элемента, не имеющего владельца (например, сразу после вызова MyControl:=TListBox.Create(...) происходит обращение к методу Refresh).

EInvalidPointer - происходит при попытке освобождения уже освобожденного или еще неинициализированного указателя, при вызове Dispose(), FreeMem() или деструктора класса.

EListError - вызывается при обращении к элементу наследника TList по индексу, выходящему за пределы допустимых значений (например, объект TStringList содержит только 10 строк, а происходит обращение к одиннадцатому).

EMathError - предок исключений, случающихся при выполнении операций с плавающей точкой.
EInvalidOp - происходит, когда математическому сопроцессору передается ошибочная инструкция. Такое исключение не будет до конца обработано, пока Вы контролируете сопроцессор напрямую из ассемблерного кода.
EOverflow - происходит как результат переполнения операций с плавающей точкой при слишком больших величинах. Соответствует RunTime Error 205.
Underflow - происходит как результат переполнения операций с плавающей точкой при слишком малых величинах. Соответствует RunTime Error 206.
EZeroDivide - вызывается в результате деления на ноль.
EMenuError - вызывается в случае любых ошибок при работе с пунктами меню для компонент TMenu, TMenuItem, TPopupMenu и их наследников.
EOutlineError - вызывается в случае любых ошибок при работе с TOutLine и любыми его наследниками.
EOutOfMemory - происходит в случае вызовов New(), GetMem() или конструкторов классов при невозможности распределения памяти. Соответствует RunTime Error 203.
EOutOfResources - происходит в том случае, когда невозможно выполнение запроса на выделение или заполнение тех или иных Windows ресурсов (например, таких, как обработчики - handles).
EParserError – вызывается, когда Delphi не может произвести разбор и перевод текста описания формы в двоичный вид (часто происходит в случае исправления текста описания формы вручную в IDE Delphi).
EPrinter - вызывается в случае любых ошибок при работе с принтером.
EProcessorException - предок исключений, вызываемых в случае прерывания процессора- hardware breakpoint. Никогда не включается в DLL, может обрабатываться только в “цельном” приложении.
EBreakpoint - вызывается в случае останова на точке прерывания при отладке в IDE Delphi. Среда Delphi обрабатывает это исключение самостоятельно.
EFault - предок исключений, вызываемых в случае невозможности обработки процессором тех или иных операций.
EGPFault - вызывается, когда происходит “общее нарушение защиты” - General Protection Fault. Соответствует RunTime Error 216.
EInvalidOpCode - вызывается, когда процессор пытается выполнить недопустимые инструкции.
EPageFault - обычно происходит как результат ошибки менеджера памяти Windows, вследствие некоторых ошибок в коде приложения. После такого исключения рекомендуется перезапустить Windows.

EStackFault - происходит при ошибках работы со стеком, часто вследствие некорректных попыток доступа к стеку из фрагментов кода на ассемблере. Компиляция программ со включенной проверкой работы со стеком {$S+} помогает отследить такого рода ошибки.
ESingleStep - аналогично EBreakpoint, это исключение происходит при пошаговом выполнении приложения в IDE Delphi, которая сама его и обрабатывает.
EPropertyError - вызывается в случае ошибок в редакторах свойств, встраиваемых в IDE Delphi. Имеет большое значение для написания надежных property editors. Определен в модуле DsgnIntf.pas.
EResNotFound - происходит в случае тех или иных проблем, имеющих место при попытке загрузки ресурсов форм - файлов .DFM в режиме дизайнера. Часто причиной таких исключений бывает нарушение соответствия между определением класса формы и ее описанием на уровне ресурса (например,вследствие изменения порядка следования полей-ссылок на компоненты, вставленные в форму в режиме дизайнера).
EStreamError - предок исключений, вызываемых при работе с потоками.
EFCreateError - происходит в случае ошибок создания потока (например, при некорректном задании файла потока).
EFilerError - вызывается при попытке вторичной регистрации уже зарегистрированного класса (компоненты). Является, также, предком специализированных обработчиков исключений, возникающих при работе с классами компонент.
EClassNotFound - обычно происходит, когда в описании класса формы удалено поле-ссылка на компоненту, вставленную в форму в режиме дизайнера. Вызывается, в отличие от EResNotFound, в RunTime.
EInvalidImage - вызывается при попытке чтения файла, не являющегося ресурсом, или разрушенного файла ресурса, специализированными функциями чтения ресурсов (например, функцией ReadComponent).
EMethodNotFound - аналогично EClassNotFound, только при несоответствии методов, связанных с теми или иными обработчиками событий.
EReadError - происходит в том случае, когда невозможно прочитать значение свойства или другого набора байт из потока (в том числе ресурса).
EFOpenError – вызывается, когда тот или иной специфированный поток не может быть открыт (например, когда поток не существует).
EStringListError - происходит при ошибках работы с объектом TStringList (кроме ошибок, обрабатываемых TListError).
Исключения, возникающие при работе с базами данных.
Delphi, обладая средствами доступа к данным, основывающимися на интерфейсе IDAPI, реализованной в виде библиотеки Borland Database Engine (BDE), включает ряд обработчиков исключительных ситуаций для регист-

рации ошибок в компонентах VCL работающим с БД. Дадим краткую характеристику основным из них:
EDatabaseError - наследник Exception ; происходит при ошибках доступа к данным в компонентах-наследниках TDataSet. Объявлено в модуле DB.
EDBEngineError - наследник EDatabaseError ; вызывается, когда происходят ошибки BDE или на сервере БД. Объявлено в модуле DB.
Особенно важны два свойства класса EDBEngineError :
Errors - список всех ошибок, находящихся в стеке ошибок BDE. Индекс первой ошибки 0;
ErrorCount - количество ошибок в стеке.
Объекты, содержащиеся в Errors, имеют тип TDBError. Доступные свойства класса TDBError:
ErrorCode - код ошибки, возвращаемый Borland Database Engine;
Category - категория ошибки, описанной в ErrorCode;
SubCode - ‘субкод’ ошибки из ErrorCode;
NativeError - ошибка, возвращаемая сервером БД. Если NativeError 0, то ошибка в ErrorCode не от сервера;
Message - сообщение, переданное сервером, если NativeError не равно 0; сообщение BDE - в противном случае.
EDBEditError - наследник Exception ; вызывается, когда данные не совместимы с маской ввода, наложенной на поле. Объявлено в модуле Mask.

Вопросы для самоподготовки:

1. Что такое исключительная ситуация?
2. Что такое обработчик исключительных ситуаций?
3. Чем конструкция try..except отличается от конструкции try..finally?
4. Какие существуют модели исключительных ситуаций?

Глава 4. События в Delphi

Одна из ключевых целей среды визуального программирования - скрыть от пользователя сложность программирования в Windows. При этом, однако, хочется, чтобы такая среда не была упрощена слишком, не до такой степени, что программисты потеряют доступ к самой операционной системе.
Программирование, ориентированное на события - неотъемлемая черта Windows. Некоторые программные среды для быстрой разработки приложений (RAD) пытаются скрыть от пользователя эту черту совсем, как будто она настолько сложна, что большинство не могут ее понять. Истина заключается в том, что событийное программирование само по себе не так уж сложно. Однако, есть некоторые особенности воплощения данной концепции в Windows, которые в некоторых ситуациях могут вызвать затруднения.

Delphi предоставляет полный доступ к подструктуре событий, предоставляемой Windows. С другой стороны, Delphi упрощает программирование обработчиков таких событий.
Объекты из библиотеки визуальных компонент (VCL) Delphi, равно как и объекты реального мира, имеют свой набор свойств и свое поведение - набор откликов на события, происходящие с ними. Список событий для данного объекта, на которые он реагирует, можно посмотреть, например, в Инспекторе Объектов на странице событий. Среди набора событий для различных объектов из VCL есть как события, портируемые из Windows (MouseMove, KeyDown), так и события, порождаемые непосредственно в программе (DataChange для TDataSource).
Поведение объекта определяется тем, какие обработчики и для каких событий он имеет. Создание приложения в Delphi состоит из настройки свойств используемых объектов и создания обработчиков событий.
Простейшие события, на которые иногда нужно реагировать - это, например, события, связанные с мышью (они есть практически у всех видимых объектов) или событие Click для кнопки TButton. Предположим, что вы хотите перехватить щелчок левой кнопки мыши на форме. Чтобы сделать это - создайте новый проект, в Инспекторе Объектов выберите страницу событий и сделайте двойной щелчок на правой части для свойства OnClick. Вы получите заготовку для обработчика данного события:

```
 procedure TForm1.FormClick(Sender: TObject);
begin
end;
```

Внутрь поместим код:

```
 procedure TForm1.FormClick(Sender: TObject);
begin
MessageDlg('Hello', mtInformation, [mbOk], 0);
end;
```

Каждый раз , когда делается щелчок левой кнопки мыши над формой будет появляться окно диалога.

Диалог, появляющийся при щелчке мыши на форме.

Код, приведенный выше, представляет собой простейший случай ответа на событие в программе на Delphi. Программисты в Windows знают, что при возникновении события, операционная система передает не только уведомление о нем, но и некоторую связанную с ним информацию. Например, при возникновении события “нажата левая кнопка мыши” про-

грамма информируется о том, в каком месте это произошло. Если вы хотите получить доступ к такой информации, то должны вернуться в Инспектор Объектов и создать обработчик события OnMouseDown.

Также можно создать обработчик для OnKeyDown (нажата клавиша).

Событийное программирование есть не только в Windows, и данную черту можно реализовать не только в операционной системе. Например, любая DOS программа может быть основана на простом цикле, работающем все время жизни программы в памяти.

Если нужно перемещать несколько картинок по экрану, то может понадобиться сдвинуть их на несколько точек, затем проверить, нажимал ли пользователь кнопки. Если такое событие было, то его можно обработать, если нет, то двигать дальше.

Одной из основных причин, почему Microsoft сделал Windows по такой схеме, является тот факт, что множество задач работает в среде одновременно. В многозадачных системах операционная система должна знать, щелкнул ли пользователь мышкой на определенное окно. Если это окно было частично перекрыто другим, то это становится известно операционной системе, и она перемещает окно на передний план. Понятно, что неудобно заставлять само окно выполнять эти действия. Операционной системе лучше обрабатывать все нажатия клавиш и кнопок на мыши и затем передавать их в остальные программы в виде событий.

Т.е., программист в Windows не должен напрямую проверять “железо”. Система выполняет эту задачу и передает информацию программе в виде сообщений.

Когда пользователь щелкает мышью, операционная система обрабатывает это событие и передает его в окно, которое должно обработать данное событие.

Каждое сообщение, посылаемое в окно, состоит из четырех частей: первая часть - handle окна, получающего сообщение, Msg сообщает, что произошло, а третья и четвертая части (wParam и lParam) содержат дополнительную информацию о событии. Вместе эти четыре части являются аналогом показанной выше структуры TEvent.

Вторая часть сообщения имеет длину 16 бит и сообщает, что за событие произошло. Например, если нажата левая кнопка на мыши, то переменная Msg содержит значение WM_LBUTTONDOWN. Существуют десятки различного типа сообщений и они называются вроде WM_GETTEXT, WM_HSCROLL, WM_GETTEXTLENGTH и т.п. Список всех сообщений можно видеть в справочнике по Windows API (on-line help).

Последние две переменные, длиной 16 и 32 бита, называются wParam и lParam. Они сообщают программисту важную дополнительную информацию о каждом событии. Например, при нажатии кнопки мыши, lParam содержит координаты указателя мыши.

Одна из хитростей заключается в том, как выделить нужную информацию из этих переменных. В большинстве случаев Delphi освобождает вас от необходимости выполнять данную задачу. Например, в обработчике события OnMouseDown для формы вы просто используете координаты X и Y. Все, что связано с событиями, представлено в простом и непосредственном виде:

```
procedure TForm1.FormMouseDown(Sender: TObject;
Button: TMouseButton;
Shift: TShiftState;
X, Y: Integer);
```

Итак, если подвести итог, то должно стать ясным следующее:

- Windows является системой ориентированной на события;
- События в Windows принимают форму сообщений;
- В недрах VCL Delphi сообщения Windows обрабатываются и преобразуются в более простую для программиста форму;
- Обработка событий в Delphi сводится к написанию для каждого объекта своих обработчиков;
- События в программе на Delphi вызываются не только сообщениями Windows, но и внутренними процессами.

Конечно, нельзя придумать такую библиотеку объектов, которые бы полностью соответствовали потребностям программистов. Всегда возникнет необходимость дополнения или изменения свойств и поведения объектов. В этом случае, так же, как и при создании своих собственных компонент в Delphi, часто требуется обрабатывать сообщения Windows. Поскольку Object Pascal является развитием и продолжением Borland Pascal 7.0, то это выполняется сходным с BP способом.

Общий синтаксис для декларации обработчика сообщений Windows:

```
procedure Handler_Name(var Msg : MessageType);
message WM_XXXXX;
```

Handler_Name обозначает имя метода; Msg - имя передаваемого параметра; MessageType - какой либо тип записи, подходящий для данного сообщения; директива message указывает, что данный метод является обработчиком сообщения; WM_XXXXX - константа или выражение, которое определяет номер обрабатываемого сообщения Windows.

Однако, есть еще способ обработки всех сообщений, которые получает приложение. Для этого используется свойство OnMessage объекта Application, который автоматически создается при запуске программы. Если определен обработчик события OnMessage, то он получает управление при любом событии, сообщение о котором направлено в программу.

Вопросы для самопроверки:

1. Что такое событие?
2. Что такое диалоговое окно?

3. Какие существуют обработчики событий?

Глава 5. Средства создания мультимедийных приложений

Delphi позволяет включать в программу такие мультимедийные объекты, как звуки, видео и музыку, используя встроенный в Delphi компонент TMediaPlayer.

Мультимедиа - это термин, относящийся к почти всем формам анимации, звукам, видео, которые используются на компьютере, которое включает:

1. Показ видео в формате Microsoft's Video for Windows (AVI).
2. Воспроизведение звуков и музыки из MIDI и WAVE файлов.

Данную задачу можно выполнить с помощью динамической библиотеки Microsoft Multimedia Extensions для Windows (MMSYSTEM.DLL), методы которой инкапсулированы в компоненте TMediaPlay, находящийся на странице System Палитры Компонент Delphi.

Для проигрывания файлов мультимедиа может потребоваться наличие некоторого оборудования и программного обеспечения. Так для воспроизведения звуков нужна звуковая карта.

Компонент TMediaPlayer оформлен, как панель управления устройством с кнопками. Как и на магнитофоне, здесь есть кнопки “воспроизведение”, “перемотка”, “запись” и др.

Существует два вида программ мультимедиа:

1. Когда приходится предоставлять пользователям простой путь для проигрывания максимально широкого круга файлов. Это означает, что Вам нужно будет дать пользователю доступ к жесткому диску или CD-ROM, и затем позволить ему выбрать и воспроизвести подходящий файл. В этом случае, на форме обычно располагается TMediaPlayer, предоставляющий возможность управления воспроизведением.
2. Когда программист может захотеть скрыть от пользователя существование компонента TMediaPlayer. То есть, воспроизвести звук или видео без того, чтобы пользователь заботился об их источнике. В частности, звук может быть частью презентации. Например, показ какого-нибудь графика на экране может сопровождаться объяснением, записанным в WAV файл. В течение презентации пользователь даже не знает о существовании TMediaPlayer. Он работает в фоновом режиме. Для этого компонент делается невидимым (Visible = False) и управляется программно.

Во время выполнения программы может потребоваться отобразить текущее состояние объекта MediaPlayer и самого ролика (время, прошедшее с начала воспроизведения, длину ролика). Для этого у объекта TMediaPlayer есть соответствующие свойства и события: Length, Position, OnNotify и др.

Для обновления показаний индикатора можно воспользоваться таймером. Для этого надо поместить на форму объект TTimer и установить для него Interval = 100 (100 миллисекунд).

Глава 6. Введение в Object Pascal

6.1 Структура программ Delphi

Любая программа в Delphi состоит из файла проекта (файл с расширением dpr) и одного или нескольких модулей (файлы с расширениями pas). Каждый из таких файлов описывает программную единицу Object Pascal.
Файл проекта представляет собой программу, написанную на языке Object Pascal и предназначенную для обработки компилятором. Эта программа автоматически создается Delphi и содержит лишь несколько строк. Чтобы увидеть их, запустите Delphi и щелкните по опции Project | View Source главного меню [В предыдущих версиях Delphi для просмотра кода проекта используйте опцию View | project Source.]. Delphi покажет окно кода с закладкой Project1, содержащее такой текст:

```
program Project1;
uses
Forms, Unit1 in 'Unit1.pas' {fmExample};
{$R *.RES}
begin
Application.Initialize;
Application.CreateForm(TfmExample, fmExample);
Application.Run;
end.
```

В окне кода жирным шрифтом выделяются так называемые зарезервированные слова, а курсивом - комментарии (так же выделяются зарезервированные слова и комментарии в книге). Как видим, текст программы начинается зарезервированным словом program и заканчивается словом end с точкой за ним. Сочетание end со следующей за ней точкой называется терминатором программной единицы: как только в тексте программы встретится такой терминатор, компилятор прекращает анализ программы и игнорирует оставшуюся часть текста.
Зарезервированные слова играют важную роль в Object Pascal, придавая программе в целом свойство текста, написанного на почти естественном английском языке. Каждое зарезервированное слово (а их в Object Pascal несколько десятков) несет в себе условное сообщение для компилятора, который анализирует текст программы так же, как читаем его и мы: слева направо и сверху вниз.

Комментарии, наоборот, ничего не значат для компилятора, и он их игнорирует. Комментарии важны для программиста, который с их помощью поясняет те или иные места программы. Наличие комментариев в тексте программы делает ее понятнее и позволяет легко вспомнить особенности реализации программы, которую вы написали несколько лет назад. В Object Pascal комментарием считается любая последовательность символов, заключенная в фигурные скобки. В приведенном выше тексте таких комментариев два, но строка

{$R *.RES}

на самом деле не является комментарием. Этот специальным образом написанный фрагмент кода называется директивой компилятора (в нашем случае - указание компилятору на необходимость подключения к программе так называемого файла ресурсов). Директивы начинаются символом $, который стоит сразу за открывающей фигурной скобкой.

В Object Pascal в качестве ограничителей комментария могут также использоваться пары символов (*, *) и //. Скобки (*...*) используются подобно фигурным скобкам, т. е. комментарием считается находящийся в них фрагмент текста, а символы // указывают компилятору, что комментарий располагается за ними и продолжается до конца текущей строки:

{Это комментарий}

(*Это тоже комментарий*)

//Все символы до конца этой строки составляют комментарий

Слово Program со следующим за ним именем программы и точкой с запятой образуют заголовок программы. За заголовком следует раздел описаний, в котором программист (или Delphi) описывает используемые в программе идентификаторы. Идентификаторы обозначают элементы программы, такие как типы, переменные, процедуры, функции (об элементах программы мы поговорим чуть позже). Здесь же с помощью предложения, которое начинается зарезервированным словом uses («использовать») программист сообщает компилятору о тех фрагментах программы (модулях), которые необходимо рассматривать как неотъемлемые составные части программы и которые располагаются в других файлах. Строки

uses

Forms, Unit1 in 'Unitl.pas' {fmExample};

указывают, что помимо файла проекта в программе должны использоваться модули Forms И Unit1. модуль Forms является стандартным (т. е. уже известным Delphi), а модуль Unit1 - новым, ранее неизвестным, и Delphi в этом случае указывает также имя файла с текстом модуля (in 'uniti.pas') и имя связанного с модулем файла описания формы {fmExample}.

Собственно тело программы начинается со слова begin (начать) и ограничивается терминатором end с точкой. Тело состоит из нескольких операторов языка Object Pascal. В каждом операторе реализуется некоторое дейст-

вие - изменение значения переменной, анализ результата вычисления, обращение к подпрограмме и т. п. В теле нашей программы - три исполняемых оператора:

```
Application.Initialize;
Application.CreateForm(TfmExample, fmExample);
Application.Run;
```

Каждый из них реализует обращение к одному из методов объекта Application. Объектом называется специальным образом оформленный фрагмент программы, заключающий в себе данные и подпрограммы для их обработки. Данные называются полями объекта, а подпрограммы - его методами. Объект в целом предназначен для решения какой-либо конкретной задачи и воспринимается в программе как неделимое целое (иными словами, нельзя из объекта “выдернуть” отдельное поле или метод). Объекты играют чрезвычайно важную роль в современных языках программирования. Они придуманы для того, чтобы увеличить производительность труда программиста и одновременно повысить качество разрабатываемых им программ. Два главных свойства объекта - функциональность и неделимость - делают его самостоятельной или даже самодостаточной частью программы и позволяют легко переносить объект из одной программы в другую. Разработчики Delphi придумали для сотни объектов, которые можно рассматривать как кирпичики, из которых программист строит многоэтажное здание программы. Такой принцип построения программ называется объектно-ориентированным программированием (ООП). В объекте Application собраны данные и подпрограммы, необходимые для нормального функционирования Windows-программы в целом. Delphi автоматически создает объект-программу Application для каждого нового проекта. Строка

```
Application.Initialize;
```

означает обращение к методу Initialize объекта Application. Прочитав эту строку, компилятор создаст код, который заставит процессор перейти к выполнению некоторого фрагмента программы, написанного для нас разработчиками Delphi. После выполнения этого фрагмента (программисты говорят: после выхода из подпрограммы) управление процессором перейдет к следующей строке программы, в которой вызывается метод CreateForm и т. д.

Модули - это программные единицы, предназначенные для размещений фрагментов программ. С помощью содержащегося в них программного кода реализуется вся поведенческая сторона программы. Любой модуль имеет следующую структуру: заголовок секция интерфейсных объявлений секция реализации терминатор Заголовок открывается зарезервированным словом Unit за которым следует имя модуля и точка с запятой. Секция интерфейсных объявлений открывается зарезервированным словом Interface,

а секция реализации - словом implementation. Терминатором модуля, как и терминатором программы, является end с точкой. Следующий фрагмент программы является синтаксически правильным вариантом модуля:

```
unit Unit1;
interface
// Секция интерфейсных объявлений
implementation
// Секция реализации
end.
```

В секции интерфейсных объявлений описываются программные элементы (типы, классы, процедуры и функции), которые будут "видны" другим программным модулям, а в секции реализации раскрывается механизм работы этих элементов. Разделение модуля на две секции обеспечивает удобный механизм обмена алгоритмами между отдельными частями одной программы. Он также реализует средство обмена программными разработками между отдельными программистами. Получив откомпилированный "посторонний" модуль, программист получает доступ только к его интерфейсной части, в которой содержатся объявления элементов. Детали реализации объявленных процедур, функций, классов скрыты в секции реализации и недоступны другим модулям.

Щелкните по закладке Unit1 окна кода, и вы увидите такой текст:

```
 unit Unit1;
interface
uses
Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs, StdCtrls, Buttons, ExtCtrls;
type
TfmExample = class(TForm)
Panel1: TPanel;
bbRun: TBitBtn;
bbClose: TBitBtn;
edinput: TEdit;
IbOutput: TLabel;
mmOutput: TMemo;
private
{ Private declarations } public
{ Public declarations } end;
var
fmExample: TfmExample;
implementation
$R *.DFM}
end.
```

Весь этот текст сформирован Delphi, но в отличие от файла проекта программист может его изменять, придавая программе нужную функциональность. В интерфейсной секции описан один тип (класс - fmExample) и один объект (переменная fmExample).
Вот описание класса:

```
type
TfmExample = class(TForm)
Panell: TPanel;
bbRun: TBitBtn;
bbClose: TBitBtn;
edinput: TEdit;
IbOutput: TLabel;
mmOutput: TMemo;
private
{ Private declarations } public
{ Public declarations } end;
```

Классы служат основным инструментом реализации мощных возможностей Delphi. Класс является образцом, по которому создаются объекты, и наоборот, объект - это экземпляр реализации класса. Образцы для создания элементов программы в Object Pascal называются типами, таким образом, класс TfmExamplel -это тип. Перед его объявлением стоит зарезервированное слово type (тип), извещающее компилятор о начале раздела описания типов.
Стандартный класс TForm реализует все нужное для создания и функционирования пустого Windows-окна. Класс TfmExamplel порожден от этого класса, о чем свидетельствует строка
TfmExample = class(TForm)
в которой за зарезервированным словом class в скобках указывается имя родительского класса. Термин "порожден" означает, что класс TfmExample унаследовал все возможности родительского класса TForm и добавил к ним собственные в виде дополнительных компонентов, которые, как вы помните, мы вставили в форму fmExample. Перечень вставленных нами компонентов и составляет значительную часть описания класса.
Свойство наследования классами-потомками всех свойств родительского класса и обогащения их новыми возможностями является одним из фундаментальных принципов объектно-ориентированного программирования. От наследника может быть порожден новый наследник, который внесет свою лепту в виде дополнительных программных заготовок и т. д. В результате создается ветвящаяся иерархия классов, на вершине которой располагается самый простой класс TObject (все остальные классы в Delphi порождены от этого единственного прародителя), а на самой нижней сту-

пени иерархии - мощные классы-потомки, которым по плечу решение любых проблем.
Объект fmExampie формально относится к элементам программы, которые называются переменными. Вот почему перед объявлением объекта стоит зарезервированное слово var (от англ. variables - переменные).
Текст модуля доступен как Delphi ,так и программисту. Delphi автоматически вставляет в текст модуля описание любого добавленного к форме компонента, а также создает заготовки для обработчиков событий; программист может добавлять свои методы в ранее объявлённые классы и наполнять обработчики событий конкретным содержанием, вставлять собственные переменные, типы, константы и т. д. Совместное с Delphi владение текстом модуля будет вполне успешным, если программист будет соблюдать простое правило, он не должен удалять или изменять строки которые вставлены не им, а Delphi.

6.2 Элементы программы

Элементы программы - это минимальные неделимые ее части, еще несущие в себе определенную значимость для компилятора. К элементам относятся:
зарезервированные слова;
идентификаторы;
типы;
константы;
переменные;
метки;
подпрограммы;
комментарии.
Зарезервированные слова это английские слова, указывающие компилятору на необходимость выполнения определенных действий. Зарезервированные слова не могут использоваться в программе ни для каких иных целей кроме тех, для которых они предназначены. Например, зарезервированное слово begin означает для компилятора начало составного оператора. Программист не может создать в программе переменную с именем begin, константу begin, метку begin или вообще какой бы то ни было другой элемент программы с именем begin.
Идентификаторы - это слова, которыми программист обозначает любой другой элемент программы, кроме зарезервированного слова, идентификатора или комментария. Идентификаторы в Object Pascal могут состоять из латинских букв, арабских цифр и знака подчеркивания. Никакие другие символы или специальные знаки не могут входить в идентификатор. Из этого простого правила следует, что идентификаторы не могут состоять из

нескольких слов (нельзя использовать пробел) или включать в себя символы кириллицы (русского алфавита).
Типы - это специальные конструкции языка, которые рассматриваются компилятором как образцы для создания других элементов программы, таких как переменные, константы и функции. Любой тип определяет две важные для компилятора вещи: объем памяти, выделяемый для размещения элемента (константы, переменной или результата, возвращаемого функцией), и набор допустимых действий, которые программист может совершать над элементами данного типа. Замечу, что любой определяемый программистом идентификатор должен быть описан в разделе описаний (перед началом исполняемых операторов). Это означает, что компилятор должен знать тот тип (образец), по которому создается определяемый идентификатором элемент.
Константы определяют области памяти, которые не могут изменять своего значения в ходе работы программы. Как и любые другие элементы программы, константы могут иметь свои собственные имена. Объявлению имен констант должно предшествовать зарезервированное слово const (от англ. constants - константы). Тип константы определяется способом ее записи и легко распознается компилятором в тексте программы, поэтому программист может не использовать именованные константы (т. е. не объявлять их в программе явно).
Переменные связаны с изменяемыми областями памяти, т. е. с такими ее участками, содержимое которых будет меняться в ходе работы программы. В отличие от констант переменные всегда объявляются в программе. Для этого после идентификатора переменной ставится двоеточие и имя типа, по образу которого должна строиться переменная. Разделу объявления переменной (переменных) должно предшествовать слово var. Например:

```
var
inValue: Integer;
byValue: Byte;
```

Здесь идентификатор inValue объявляется как переменная типа integer, а идентификатор byValue - как переменная типа Byte. Стандартный (т. е. заранее определенный в Object Pascal) тип integer определяет четырехбайтный участок памяти, содержимое которого рассматривается как целое число в диапазоне от -2 147 483 648 до+2 147 483 647, а стандартный тип Byte - участок памяти длиной 1 байт, в котором размещается беззнаковое целое число в диапазоне от 0 до 2554.
Метки - это имена операторов программы. Метки используются очень редко и только для того, чтобы программист смог указать компилятору, какой оператор программы должен выполнятся следующим. Метки, как и переменные, всегда объявляются в программе. Разделу объявлений меток предшествует зарезервированное слово label (метка). Например:

```
label
Loop;
begin
Goto Loop;
// Программист требует передать управление оператору, помеченному мет-
кой Loop.
 .....
// Эти операторы будут пропущены
Loop:
// Оператору, идущему за этой меткой,
.....
// будет передано управление
end;
```

Подпрограммы - это специальным образом оформленные фрагменты программы. Замечательной особенностью подпрограмм является их значительная независимость от остального текста программы. Говорят, что свойства подпрограммы локализуются в ее теле. Это означает, что, если программист что-либо изменит в подпрограмме, ему, как правило, не понадобится в связи с этим изменять что-либо вне подпрограммы. Таким образом, подпрограммы являются средством структурирования программ, т. е. расчленения программ на ряд во многом независимых фрагментов. Структурирование неизбежно для крупных программных проектов, поэтому подпрограммы используются в Delphi-программах очень часто.

В Object Pascal есть два сорта подпрограмм: процедуры и функции. Функция отличается от процедуры только тем, что ее идентификатор можно наряду с константами и переменными использовать в выражениях, т. к. функция имеет выходной результат определенного типа. Если, например, определена функция

```
Function MyFunction: Integer;
```

и переменная var

```
X: Integer;
```

то возможен такой оператор присваивания:

```
X := 2*MyFunction-l;
```

Имя процедуры нельзя использовать в выражении, т. к. процедура не имеет связанного с нею результата:

```
Procedure MyProcedure;
…
X := 2*MyProcedure-l; // Ошибка!
```

<u>Типы.</u>

Типы в Object Pascal играют огромную роль. С их помощью определяются классы - основной инструмент программиста.

<u>Строковый и символьный типы.</u>

Строковый тип String. Этот тип определяет участок памяти переменной длины, каждый байт которого содержит один символ. Для символов в Object Pascal используется тип Char, таким образом, String - это цепочка следующих друг за другом символов Char. Каждый символ в String пронумерован, причем первый символ имеет номер 1. Программист может обращаться к любому символу строки, указывая его порядковый номер в квадратных скобках сразу за именем переменной:

```
var // Начало раздела описания переменных
S: String;
// Объявление переменной строкового типа
begin
// Начало раздела исполняемых операторов
S := 'Строка символов';
// Переменная S содержит
// значение ”Строка символов”
S[6] := 'и'; // Теперь переменная содержит значение
// ”Строки символов”
end;
// Конец раздела исполняемых операторов
```

В первом операторе присваивания в переменную s будет помещено значение строковой константы строка символов'. Строковые константы содержат произвольные символы, заключенные в обрамляющие апострофы, причем сами апострофы не входят в значение константы, поэтому после присваивания переменная примет значение Строка символов без апострофов (если понадобится включить в текстовую константу апостроф, он удваивается: ' символ ' ' - апостроф'). После первого присваивания s будет занимать участок памяти длиной 15 байт - по одному байту на каждый символ значения. Переменность размера области памяти, выделяемой для размещения строки символов, - характерная особенность типа string. Если бы, например, во втором операторе мы обратились не к 6-му по счету символу, а ко всей строке в целом:

```
S := 'и';
```

эта переменная стала бы занимать 1 байт, а следующие за ним 14 байтов оказались бы свободными. Длина строковой переменной в программе меняется автоматически при каждом присваивании переменной нового значения и может составлять от 0 до 2 Гбайт. Над строковым типом определена операция сцепления (+):

```
S := 'Object'+' Pascal';// S содержит “Object Pascal”
```

Кроме того, строки можно сравнивать с помощью таких операций отношения:

Операция	Смысл
=	Равно

<>	Не равно
>	Больше
>=	Больше или равно
<	Меньше
<=	Меньше или равно

Примечание. Символы <>, <= и >= пишутся слитно, их нельзя разделять пробелами или комментариями.

Результат применения операции сравнения к двум строкам имеет логический тип, который характеризуется двумя возможными значениям: True (Истина) и False (Ложь). Строки сравниваются побайтно, слева направо; каждая пара символов сравнивается в соответствии с их внутренней кодировкой.

Все остальные действия над строками осуществляются с помощью нескольких стандартных для Delphi подпрограмм.

Целые типы.

Целые типы используются для хранения и преобразования целых чисел. В Object Pascal предусмотрено несколько целочисленных типов, отличающихся диапазоном возможных значений. Над целыми числами определены следующие математические операции:

+ сложение
- вычитание
* умножение
div деление с отбрасыванием остатка
mod получение остатка от деления

Спецификой деления является то обстоятельство, что результат может иметь дробный вид: 1/2, 2*3/5 и т. п. Для хранения дробных чисел в Object Pascal используются вещественные типы, вот почему в языке имеются целых две операции деления (div и mod):

```
var
X,Y: Integer;
begin
X := 5 div 2; // X содержит 2
Y := 5 mod 2; // Y содержит 1
end;
```

Как и к строкам, к целым числам применимы операции сравнения.

6.3 Операторы языка

Составной оператор и пустой оператор.

Составной оператор - это последовательность произвольных операторов программы, заключенная в операторные скобки - зарезервированные слова begin ... end. Составные операторы - важный инструмент Object Pascal,

дающий возможность писать программы по современной технологии структурного программирования (без операторов перехода goto).

Object Pascal не накладывает никаких ограничений на характер операторов, входящих в составной оператор. Среди них могут быть и другие составные операторы - язык Object Pascal допускает произвольную глубину их вложенности:

Фактически весь раздел операторов, обрамленный словами begin ... end, представляет собой один составной оператор. Поскольку зарезервированное слово end является закрывающей операторной скобкой, оно одновременно указывает и конец предыдущего оператора, поэтому ставить перед ним символ “;” необязательно. Наличие точки с запятой перед end в предыдущих примерах означало, что между последним оператором и операторной скобкой end располагается пустой оператор. Пустой оператор не содержит никаких действий, просто в программу добавляется лишняя точка с запятой. В основном пустой оператор используется для передачи управления в конец составного оператора: как и любой другой, пустой оператор может быть помечен, и ему можно передать управление.

Условный оператор

Условный оператор позволяет проверить некоторое условие и в зависимости от результатов проверки выполнить то или иное действие. Таким образом, условный оператор - это средство ветвления вычислительного процесса.

Структура условного оператора имеет следующий вид:

if <условие> then <оператор1> else <оператор2>;

где if/ then/ else - зарезервированные слова (если, то, иначе);

<условие> - произвольное выражение логического типа;

<оператор1>, <оператор2> - любые операторы языка Object Pascal.

Условный оператор работает по следующему алгоритму. Вначале вычисляется условное выражение <условие>. Если результат есть True (истина), то выполняется <оператор1>, а <оператор2> пропускается; если результат есть False (ложь), наоборот, <оператор1> пропускается, а выполняется <оператор2>.

Условными называются выражения, имеющие одно из двух возможных значений: истина или ложь. Такие выражения чаще всего получаются при сравнении переменных с помощью операций отношения =, <>, >, >=, <, <=. Сложные логические выражения составляются с использованием логических операций and (логическое И), or (логическое ИЛИ) и not (логическое НЕ). Например:

if (a > b) and (b <> 0) then ...

В отличие от других языков программирования в Object Pascal приоритет операций отношения меньше, чем у логических операций, поэтому от-

дельные составные части сложного логического выражёния заключаются в скобки. Например, такая запись предыдущего оператора будет неверной:
if a>b and b <> 0 then ...// Ошибка, так как фактически (с учетом приоритета операции) компилятор будет транслировать такую строку:
if a>(b and b)<>0 then...
Часть else <оператор2> условного оператора может быть опущена. Тогда при значении True условного выражения выполняется <оператор1>, в противном случае этот оператор пропускается.
Поскольку любой из операторов <оператор1> и <оператор2> может быть любого типа, в том числе и условным, а в то же время не каждый из “вложенных” условных операторов может иметь часть else <оператор2>, то возникает неоднозначность трактовки условий. Эта неоднозначность в Object Pascal решается следующим образом:
- любая встретившаяся часть else соответствует ближайшей к ней сверху по тексту программы части then условного оператора.

Операторы повторений.

В языке Object Pascal имеются три различных оператора, с помощью которых можно запрограммировать повторяющиеся фрагменты программ.
Счетный оператор цикла FOR имеет такую структуру:
for <параметр цикла> := <нач_знач> to <кон знач> do <оператор>;
Здесь for, to, do - зарезервированные слова (для, до, выполнить);
<параметр_цикла> - переменная типа Integer (точнее, любого порядкового типа); <нач_знач> - начальное значение - выражение того же типа; <кон_знач> - конечное значение - выражение того же типа; <оператор> - произвольный оператор Object Pascal.
При выполнении оператора for вначале вычисляется выражение <нач_знач> и осуществляется присваивание <параметр_цикла> := <нач_знач>. После этого циклически повторяется:
проверка условия <параметр_цикла> <= <кон_знач>; если условие не выполнено, оператор for завершает свою работу;
выполнение оператора <оператор>;
наращивание переменной <параметр_цикла> на единицу.
С помощью зарезервированных слов try (попробовать), except (исключение) и end реализуется так называемый защищенный блок. Такими блоками программист может защитить программу от краха при выполнении потенциально опасного. Если обнаружена ошибка, возникает так называемая исключительная ситуация (исключение) и управление автоматически передается оператору, стоящему за except, - начинается обработка исключения. Вначале с помощью стандартной процедуры ShowMessage мы сообщаем пользователю об ошибке. Отметим также два обстоятельства. Во-первых, условие, управляющее работой оператора for, проверяется перед выполнением оператора <оператор>: если условие не выполняется в самом

начале работы оператора for, исполняемый оператор не будет выполнен ни разу. Другое обстоятельство - шаг наращивания параметра цикла строго постоянен и равен (+1). Существует другая форма оператора:
for <пар_цик>: = <нач_знач>downto <кон_знач>do <оператор>;
Замена зарезервированного слова to на downto означает, что шаг наращивания параметра цикла равен (-1), а управляющее условие Приобретает вид <параметр__цикла> = <кон_знач>.
Оператор цикла WHILE с предпроверкой условия:
while <условие> do <оператор>;
Здесь while, do - зарезервированные слова {пока [выполняется условие], делать), <условие> - выражение логического типа; <оператор> - произвольный оператор Object Pascal.
Если выражение <условие> имеет значение True, то выполняется <оператор>, после чего вычисление выражения <условие> и его проверка повторяются. Если <условие> имеет значение False, оператор while прекращает свою работу.
Оператор цикла REPEAT... UNTIL с постпроверкой условия:
repeat <тело цикла> Until <условие>;
Здесь repeat, until - зарезервированные слова (повторять [до тех пор}, пока [не будет выполнено условие]); <тело_цикла> - произвольная последовательность операторов Object Pascal; <условие> - выражение логического типа.
Операторы <тело_цикла> выполняются хотя бы один раз, после чего вычисляется выражение <условие>: если его значение есть False, операторы <тело_цикла> повторяются, в противном случае оператор repeat... until завершает свою работу.
Обратите внимание: пара repeat... unti1 подобна операторным скобкам begin ... end, поэтому перед until ставить точку с запятой необязательно.
Замечу, что для правильного выхода из цикла условие выхода должно меняться внутри операторов, составляющих тело цикла while или repeat... until.
Для гибкого управления циклическими операторами for, while и repeat в состав Object Pascal включены две процедуры без параметров:
break - реализует немедленный выход из цикла; действие процедуры заключается в передаче управления оператору, стоящему сразу за концом циклического оператора;
continue - обеспечивает досрочное завершение очередного прохода цикла; эквивалент передачи управления в самый конец циклического оператора.
Введение в язык этих процедур практически исключает необходимость использования операторов безусловного перехода.
Оператор выбора.

Оператор выбора позволяет выбрать одно из нескольких возможных продолжений программы. Параметром, по которому осуществляется выбор, служит ключ выбора - выражение любого порядкового типа.
Структура оператора выбора такова:
case <ключ_выбора> of <список_выбора> [else <операторы>] end;
Здесь case, of, else, end - зарезервированные слова (случай, из, иначе, конец); <ключ_выбора> - ключ выбора (выражение порядкового типа); <список_выбора> - одна или более конструкций вида:
<константа_выбора> : <оператор>;
<константа_выбора> - константа того же типа, что и выражение <ключ_выбора>; <оператор> - произвольный оператор Object Pascal.
Оператор выбора работает следующим образом. Вначале вычисляется значение выражения <ключ_выбора>, а затем в последовательности операторов <список_выбора> отыскивается такой, которому предшествует константа, равная вычисленному значению. Найденный оператор выполняется, после чего оператор выбора завершает свою работу. Если в списке выбора не будет найдена константа, соответствующая вычисленному значению ключа выбора, управление передается операторам, стоящим за словом else. Часть else <операторы> можно опускать. Тогда при отсутствии в списке выбора нужной константы ничего не произойдет, и оператор выбора просто завершит свою работу.
Любому из операторов списка выбора может предшествовать не одна, а несколько констант выбора, разделенных запятыми.

Метки и операторы перехода.

Можно теоретически показать, что рассмотренных операторов вполне достаточно для написания программ любой сложности. В этом отношении наличие в языке операторов перехода кажется излишним. Более того, современная технология структурного программирования основана на принципе “программировать без GOTO”: считается, что злоупотребление операторами перехода затрудняет понимание программы, делает ее запутанной и сложной в отладке. Тем не менее, в некоторых случаях использование операторов перехода может упростить программу.
Оператор перехода имеет вид:
goto <метка>;
Здесь goto - зарезервированное слово (перейти [на метку]); <метка> - метка.
Метка в Object Pascal - это произвольный идентификатор, позволяющий именовать некоторый оператор программы и таким образом ссылаться на него. В целях совместимости со стандартным языком Паскаль в Object Pascal допускается в качестве меток использование также целых чисел без знака.

Метка располагается непосредственно перед помечаемым оператором и отделяется от него двоеточием. Оператор можно помечать несколькими метками, которые в этом случае отделяются друг от друга двоеточием. Перед тем как появиться в программе, метка должна быть описана. Описание меток состоит из зарезервированного слова label (метка), за которым следует список меток.
Действие оператора goto состоит в передаче управления соответствующему помеченному оператору.
При использовании меток необходимо руководствоваться следующими правилами:
-метка, на которую ссылается оператор goto, должна быть описана в разделе описаний, и она обязательно должна встретиться где-нибудь в теле программы;
-метки, описанные в подпрограмме, локализуются в ней, поэтому передача управления извне подпрограммы на метку внутри нее невозможна.
Массивы.
Рассмотренные выше простые типы данных позволяют использовать в программе одиночные объекты - числа, символы, строки и т. п. В Object Pascal могут использоваться также объекты, содержащие множество однотипных элементов. Это массивы - формальное объединение нескольких однотипных объектов (чисел, символов, строк и т. п.), рассматриваемое как единое целое. К необходимости применения массивов мы приходим всякий раз, когда требуется связать и использовать целый ряд родственных величин. Например, результаты многократных замеров температуры воздуха в течение года удобно рассматривать как совокупность вещественных чисел, объединенных в один сложный объект - массив измерений.
При описании массива необходимо указать общее количество входящих в массив элементов и тип этих элементов. Например:

```
var
a : array [1..10] of Real;
b : array [0..50] of Char;
c : array [-3..4] of String;
```

При описании массива используются зарезервированные слова array и of (массив, из). За словом array в квадратных скобках указывается тип-диапазон, с помощью которого компилятор определяет общее количество элементов массива. Тип-диапазон задается левой и правой границами изменения индекса массива, так что массив A состоит из 10 элементов, массив B - из 51, а массив C - из 8 элементов. За словом of указывается тип элементов, образующих массив.
Доступ к каждому элементу массива в программе осуществляется с помощью индекса - целого числа, служащего своеобразным именем элемента в массиве (если левая граница типа-диапазона равна 1, индекс элемента сов-

падает с его порядковым номером). При упоминании в программе любого элемента массива сразу за именем массива должен следовать индекс элемента в квадратных скобках.
Индекс не должен выходить за пределы, определенные типом-диапазоном. Например, можно использовать элементы а [1], в[38], с[0], но нельзя а [0] или с[38] (определение массивов см. выше). Компилятор Object Pascal может контролировать использование индексов в программе, как на этапе компиляции, так и на этапе прогона программы.
Процедуры и функции.
Процедуры и функции (их общее название – подпрограммы) представляют собой важный инструмент Object Pascal, позволяющий писать хорошо структурированные программы. В структурированных программах обычно легко прослеживается основной алгоритм, их проще понять, они удобнее в отладке и менее чувствительны к ошибкам программирования. Все эти свойства являются следствием важной особенности подпрограмм, каждая из которых представляет собой во многом самостоятельный фрагмент программы, связанный с основной программой лишь с помощью нескольких параметров. Самостоятельность подпрограмм позволяет локализовать в них все детали программной реализации того или иного алгоритмического действия, и поэтому изменение этих деталей, например, в процессе отладки обычно не приводит к изменениям основной программы.
Практически во всех языках программирования имеются средства структурирования. Языки, в которых предусмотрены такие механизмы, называются процедурно-ориентированными. К их числу принадлежит и Object Pascal.
Процедурой в Object Pascal называется особым образом оформленный фрагмент программы, имеющий собственное имя. Упоминание этого имени в тексте программы приводит к активизации процедуры и называется ее вызовом. Сразу после активизации процедуры начинают выполняться входящие в нее операторы, после выполнения последнего из них управление возвращается обратно в основную программу и выполняются операторы, стоящие непосредственно за оператором вызова.
Для обмена информацией между основной программой и процедурой используется один или несколько параметров вызова. Процедуры могут иметь и другой механизм обмена данными с вызывающей программой, так что параметры вызова могут и не использоваться. Если они есть, то они перечисляются в круглых скобках за именем процедуры и вместе с ним образуют оператор вызова процедуры.
Функция отличается от процедуры тем, что результат ее работы возвращается в виде значения этой функции, и, следовательно, вызов функции может использоваться наряду с другими операндами в выражениях.

Стандартные процедуры - Exit, ShowMessage, функции StrToInt, FioatToStr, Random, математические функции и др. Стандартными они называются потому, что созданы одновременно с системой Delphi и являются ее неотъемлемой частью. В Delphi имеется много стандартных процедур и функций. Наличие богатой библиотеки таких программных заготовок существенно облегчает разработку прикладных программ. Однако в большинстве случаев некоторые специфичные для данной прикладной программы действия не находят прямых аналогов в библиотеках Delphi, и тогда программисту приходится разрабатывать свои, нестандартные процедуры и функции.
Нестандартную подпрограмму необходимо описать, чтобы компилятор смог установить связь между оператором вызова и теми действиями, которые предусмотрены в подпрограмме. Описание подпрограммы помещается в разделе описаний (до начала исполняемых операторов).
Описание процедуры начинается зарезервированным словом procedure, за которым следуют имя процедуры и список формальных параметров. Список параметров заключается в круглые скобки и содержит перечень параметров с указанием их типа. Заметим, что перед параметром stout, с помощью которого в вызывающую программу возвращается результат преобразования, стоит зарезервированное слово var. Именно таким способом компилятору указываются те параметры, в которых процедура возвращает вызвавшей ее программе результат своей работы. Зарезервированное слово procedure, имя процедуры и список ее параметров образуют заголовок процедуры. За заголовком следует тело процедуры, содержащее новый раздел описаний.

Глава 7. Формы

Форма является основным строительным блоком в Delphi. Любая программа имеет как минимум одну связанную с ней форму, которая называется главной, - эта форма появляется на экране в момент старта программы. Однако программа может иметь сколько угодно форм, каждая из которых решает какую-то локальную задачу и появляется на экране по мере надобности. В этом разделе мы познакомимся с назначением и способами использования различных форм и изучим их свойства и методы.

7.1 Разновидности форм

Разновидности форм определяются значениями их свойств FormStyle, а также разнообразием форм-заготовок, хранящихся в репозитории Delphi.

Стиль формы задается одним из значений свойства

```
TFormStyle = (fsNormal, fsMDIChild, fsMDIForm, fsStayOnTop) ;
property FormStyle: TFormStyle;
```

Стиль fsNormal определяет обычную форму, использующуюся для решения самых различных задач, в том числе - для общего управления всей программой (главная форма).

Стили fsMDIChild и fsMDIForm используются при создании так называемых многодокументных приложений в стиле MDI (MDI -Multi Document Interface). Этот немодный сегодня стиль предполагает создание главного окна MDI (его обычно называют рамочным), внутри которого по мере надобности появляются дочерние окна. Дочерние окна, подобно дочерним элементам контейнера, не могут выходить за границы своего владельца - рамочного окна. В MDI-приложениях есть специальные средства управления взаимодействием рамочного окна с дочерними окнами. Например, каждое дочернее окно в момент активизации может нужным образом настроить главное меню рамочного окна (дочерние MDI-окна не имеют собственного главного меню). В Delphi для создания рамочного окна используется стиль fsMDIForm, а для создания дочернего MDI-окна - стиль fsMDIChild.

Стиль fsStayOnTop предназначен для окон, которые всегда должны располагаться над всеми другими окнами программы. В момент активизации окна оно обычно становится видимым на экране, даже если перед этим его загораживали другие раскрытые окна. Стиль fsStayOnTop препятствует перекрытию окна другими окнами, даже если оно становится неактивным и теряет фокус ввода. Этот стиль используется в исключительных случаях, когда окно содержит что-то, требующее повышенного внимания пользователя.

Помимо универсальной пустой формы Form существуют следующие специализированные формы:

Название	Страница	Назначение
About box	Forms	Окно О программе
Dual list box	Forms	Диалоговое окно с двумя компонентами ListBox. Используется для гибкого управления списками, в том числе для перемещения элементов из одного списка в другой
Quick Report Labels	Forms	Используется в приложениях баз данных для печати этикеток
Quick Report List	Forms	Используется в приложениях баз данных для создания обычных отчетов

Quick Report Master/Detail	Forms	Используется в приложениях баз данных для создания отчетов типа главный/детальный
Tabbed Pages	Forms	Заготовка для многостраничного диалогового окна с закладками, кнопками ok, cancel и Help
Dialog with Help	Dialogs	Заготовка для диалогового окна с кнопками ok. Cancel, Help. Имеются варианты с вертикальным асположением кнопок и с горизонтальным расположением
Password Dialog	Dialogs	Диалоговое окно с редактором TEdit, кнопками ok и Cancel для ввода паролей
Reconcile Error Dialog	Dialogs	Используется в приложениях баз данных для пояснения обнаруженной ошибки при изменении таблицы
Standard Dialog	Dialogs	Заготовка для диалогового окна с кнопками ok, cancel. Имеются варианты с вертикальным расположение кнопок и с горизонтальным расположением
Dialog Wizard	Dialogs	Мастер создания диалоговых окон
Decision Cube Sample	Business	Заготовка для использование компонентов страницы Decision Cube
Database Form Wizard	Business	Мастер создания форм для доступа к базам данных
Quick Report Wizard	Business	Мастер создания отчетов для баз данных
TeeChart Wizard	Business	Мастер форм для доступа к компоненту chart

Свойства формы:

property Active: Boolean- property ActiveControl: TWin-Control;	Содержит True, если окно активно (имеет фокус ввода) Определяет дочерний элемент, содержащий фокус ввода
property ActiveMDIChild: TForm;	Определяет дочернее midi окно с фокусом ввода
TBorderIcon = (biSystemMenu, biMinimize, biMaximize, biHelp) ; TBorderIcons = set of TBorderI-con; property Borderlcons: TBor-derIcons;	Определяет наличие кнопок в заголовке окна: biSystemMenu - имеется кнопка вызова системного меню; biMinimize - имеется кнопка минимизации; biMaximize - имеется кнопка максимизации; biHelp - имеется кнопка вызова справочной службы.
property Canvas: TCanvas;	Канва для прорисовки фона окна. Это свойство могут использовать не оконные дочерние элемента
property ClientHeight: Integer;	Высота клиентской части окна
property ClientRect: TRect;	Прямоугольник клиентской части окна
property ClientWidth: Integer;	Ширина клиентской части окна
property HelpFile: Strings;	Каждая форма может иметь индивидуальный Help- файл, имя которого содержит это свойство. Если имя не указано, используется Hdp-файл приложения
property Icon: TIcon;	Содержит пиктограмму окна. Для главной формы это свойство определяет также пиктограмму программы
property KeyPreview: Boolean;	Если имеет значение True, форма получает события от клавиатуры, перед тем как они поступят в элемент с фокусом ввода
property MDIChildCount: Integer;	В рамочном MDI-окне указывает количество связанных с ним дочерних midi-окно

property MDIChildren[I: Integer]: Tforms;	В рамочном MDI-окне открывает доступ к I-му дочернему окну
property Menu: TMainMenu;	Содержит главное меню окна
TModalResuit = Low(Integer)..High(Integer) ; property ModalResult: TModa-lResuit;	Для модального окна содержит результат диалога
property PixelsPerInch: Integer;	Определяет разрешающую способность окна в пикселях на один линейный дюйм для этапа конструирования формы
TPosition = (poDesigned, poDe-fault, poDefaultPosOnly, poDe-faultSizeOnly, poScreenCenter) ; property Position: TPosition;	Определяет положение и размеры окна в момент его появления на экране: poDesigned - такие же, как на этапе кон-струирования окна; poDefault - положе-ние и размеры определяет Windows; poDefaultposOnly - положение, как на этапе конструирования, размеры опреде-ляет Windows; л poDefaultSizeOnly - раз-меры, как на этапе конструирования, по-ложение определяет Windows; poScreenCenter - в центре экрана с разме-рами, как на этапе конструирования
TPrintScale = (poNone, poProportional, poPrintToFit); property PrintScale: TPrintScale;	Определяет масштабирование окна при его печати на принтере: poNone - нет масштабирования; каждый пиксель окна воспроизводится одной точкой на бума-ге; poproportional - форма масштабирует-ся так, чтобы ее образ на бумаге был максимально похож на ее изображение на экране; poPrintToFit- форма печатает-ся с такими же пропорциями, как на эк-ране, но с размерами, за полняющими лист бумаги
property Scaled: Boolean;	Разрешает/запрещает масштабировать форму, если значение ее свойства PixelPerinch отличается от текущего раз-решения экрана

TileMode = (tbHorizonial, tbVertical); property TileMode: TTileMode;	Определяет стиль расположения дочерних окон MDI-приложения при их упорядочении мозаикой
property WindowMenu: TMenuItem;	Определяет пункт главного меню рамочного MDI-окна, к которому добавляются опции дочернего окна
TWindowState = (wsNormal, wsMinimized, wsMaximized) ;	Определяет состояние окна в момент его появления на экране: wsNormal - бычное
property WindowState: TWindow State;	Окно; wsMinimized - минимизировано до пиктограммы; wsMaximized - распахнуто на весь экран

Если в множестве свойства BorderIcon убрать кнопки biMinimize: biMaximize, а в свойство WindowState поместить значение wsMaximized, форма займет все пространство экрана, включая нижнюю панель задач.

Методы формы:

procedure Arrangelcons;	Упорядочивает пиктограммы закрытых дочерних окон MDI-приложения
procedure Cascade;	Располагает дочерние MDI-окна каскадом
procedure Close;	Закрывает окно. Для главного окна завершает работу программы
function CloseQuery: Boolean	Возвращает True, если можно закрыть окно
procedure DefocusControl(Control: TWinControl; Removing: Boolean);	Отбирает фокус ввода у дочернего элемента Control. Если при этом Removing=True, фокус ввода получает форма
procedure FocusControl(Control: TWinControl);	Передает фокус ввода дочернему элементу Control
function GetFormImage: TBitmap;	Содержит текущее изображение окна формы
procedure Next;	Делает активным следующее mdi-okно
procedure Previous;	Делает активным предыдущее mdi-okно
procedure Tile;	Располагает дочерние MDI-окна мозаикой

procedure Print;	Печатает окно на принтере
procedure Release;	Ожидает окончания обработки всех событий формы и ее дочерних элементов, после чего уничтожает окно и освобождает всю связанную с ним память
procedure SendCancel-Mode(Sender: TControl);	Восстанавливает начальное состояние окна: освобождает мышь, прекращает прокрутку и закрывает меню
procedure SetFocus;	Передает фокус ввода форме. Форма при этом должна быть активной и видимой
procedure Show;	Показывает форму в немодальном режиме
function ShowModal: Integer;	Показывает форму в модальном режиме и возвращает результат диалога

События формы:

property OnActivate: TNoti fyEvent;	Возникает в момент активизации окна (при получениииею фокуса ввода)
TCloseAction = (caNone, caHide, caFree, caMinimize); TCloseEvent = procedure(Sender: TObject; var Action: TCloseAction) of object; property OnClose: TCloseEvent;	Возникает перед закрытием окна. Параметр Action уточняет необходимые действия: caNone - не закрывать окно; caHide - спрятать окно; caFree - уничтожить окно; caMimimize - минимизировать окно
TCloseQueryEvent = proce dure(Sender: TObject; var CanClose: Boolean) of object; property OnCloseQuery: TCloseQueryEvent;	Возникает перед закрытием окна. В параметре canclose обработчик сообщает о возможности закрытия окна
property OnCreate: TNotifyEvent;	Возникает при создании окна, но до его появления на экране
property OnDeactivate: TNoti fyEvent;	Возникает при передаче активности другому окну той же программы
property OnDestroy: TNotifyEvent;	Возникает перед разрушением окна

THelpEvent = function (Command: Word; Data: Longint; var CallHelp: Boolean): Boolean of object; property OnHelp: THelpEvent;	Возникает при обращении к справочной службе. В параметре callHelp обработчик сообщает о возможности получения помощи
property OnHide: TNotifyEvent;	Возникает перед исчезновением окна
property OnPaint: TNotifyEvent;	Возникает при необходимости прорисовки окна
property OnResize: TNotifyEvent;	Возникает при изменении размеров окна
property OnShow: TNotifyEvent;	Возникает при появлении окна на экране

7.2 Создание и использование форм

Для подключения новой формы к проекту достаточно обратиться к репозиторию и выбрать нужную разновидность формы. Менеджер проекта автоматически подключает новую форму к списку используемых форм и обеспечивает все необходимые действия по ее инициации. Самая первая подключенная к проекту форма (стандартное имя формы - Form1) становится главным окном программы. Окно этой формы автоматически появляется на экране в момент старта программы. Впрочем, программист может указать любую форму, окно которой станет главным. Для этого нужно обратиться к опции
Project | Options И, раскрыв список Main form, выбрать нужную
форму.

Каждое следующее окно становится видно только после обращения к его методу show или showModal. Чтобы обратиться к этим методам, нужно сослаться на объект-окно, который автоматически объявляется в интерфейсном разделе связанного с окном модуля. Для этого, в свою очередь, главное окно должно знать о существовании другого окна, что достигается ссылкой на модуль окна в предложении uses. Если, например, в ходе выполнения одного из методов главного окна программист захочет вызвать окно с именем fmForm, связанное с модулем Formunit, он должен сослаться на этот модуль в предложении uses главного окна:

```
implementation Uses FormUnit;
```

после чего вызвать окно на экран:

```
fmForm.Show;
```

или

```
fmForm.ShowModai;
```

Delphi автоматизирует вставку ссылки на модуль в предложение uses. Для этого на этапе конструирования нужно активизировать главное окно, щелкнув по нему мышью, после чего обратиться к опции File | uses unit. В появившемся диалоговом окне нужно выбрать модуль и нажать ок. Вставляется ссылка в предложение, стоящее за зарезервированным словом Implementation, т. к. обычно главное окно в своей интерфейсной части не ссылается на элементы интерфейсной части второго окна. Точно так же можно при необходимости сослаться в модуле второго окна на модуль главного окна: активизируйте второе окно и вновь вызовите опцию File | uses Unit. Если программист забыл сослаться на модуль, который подключен к проекту, Delphi при первой же трансляции программы сообщит об этом и предложит вставить недостающую ссылку.
При вызове метода show второе окно появляется на экране и работает одновременно с первым, поэтому управление сразу передается оператору, стоящему за обращением к этому методу. Такие окна называются немодальными, они всегда открываются в одном методе, а закрываются в другом. В отличие от этого обращение к show-Modal создает модальное окно, которое полностью берет на себя дальнейшее управление программой, поэтому оператор за обращением к showModal в вызывающей части программы получит управление только после закрытия модального окна.
Модальные окна всегда требуют от пользователя принятия какого-либо решения. С их помощью реализуется диалог с пользователем или создается информационное окно, которое пользователь должен закрыть после ознакомления с содержащейся в нем информацией. Если от пользователя требуется принятие решения, в модальное окно вставляются зависимые или независимые переключатели, кнопки и другие интерфейсные элементы, с помощью которых пользователь сможет сообщить программе о принятом решении. В момент закрытия диалога модальное окно должно поместить число, соответствующее решению пользователя, в свое свойство ModalResuit. Некоторые стандартные кнопки (ok, Yes, No, cancel и т. п.) автоматически выполняют эти действия: помещают нужное число в ModalResuit и закрывают окно. В других случаях об этом должен позаботиться программист. Вызывающая программа получает значение ModaiResuit как значение функции showModal и может тут же его проанализировать:

```
if Form2.ShowModai = mrXXX then ....
```

Возможен и такой вариант:

```
Form2.ShowModal ;
if Form2.ModaiResuit = mrXXX then .....
```

Для закрытия окна (модального или немодального) используются методы Hide или close. Следует учесть, что метод Close всегда помещает в ModalResult значение 2 (mrCancel), в то время как Hide нет меняет значения

этого свойства, поэтому, если программист хочет передать в вызывающую программу нестандартный модальный результат, следует писать:

```
ModaiResuit := MyResult;
Hide; // Но ни в коем случае Close!
```

Программы с многими формами.

Сложные программы подчас требуют создания множества форм, каждая из которых решает ту или иную конкретную задачу. Например, при создании систем управления базами данных типичным для проекта будет разработка от 50 до 100 и более форм. Процесс создания такого проекта обычно растягивается на несколько месяцев, а над его реализацией трудятся одновременно несколько программистов. Все это затрудняет ориентацию программиста, его способность быстро вспомнить назначение той или иной формы. В этом случае существенную помощь может оказать файл проекта. Рядом с описанием включенного в проекта модуля содержится строка комментария, в которой Delphi указывает имя файла формы. Этот комментарий появляется в диалоговом окне после щелчка по инструментальной кнопке выбора формы или выбора опции view | Forms главного меню. Если вы поместите в этом комментарии произвольный текст, он также будет виден в окне и поможет вам вспомнить назначение конкретной формы. Имя файла должно отделяться от вашего комментария только пробелами. Не пользуйтесь для разделения клавишей табуляции, так как в этом случае Delphi откажется включить комментарий в список форм проекта и вы просто не увидите его в диалоговом окне.

Глава 8. Среда разработчика

8.1 Главное меню

Опция File

New	**Опция-заголовок. При ее выборе раскрывается подменю со следующими опциями**
Application	Создает новую программу для Windows
CLX Application	Создает новую программу для Windows и Linux
Data Module	Создает новый модуль данных
Form	Создает новую форму и подключает ее к проекту
Frame	Создает новую раму
Other	Открывает окно репозитория

Exit	Вставляет в текущую форму ссылку на другой модуль Печатает активную форму или модуль Прекращает работу Delphi
Print	Закрывает все открытые файлы
Unit	Закрывает текущую форму
Open	Открывает ранее созданную форму
Open Project	Открывает ранее созданный проект
Reopen	Вызывает список ранее загружавшихся проектов и форм для выбора и повторной загрузки
Save Save As	Сохраняет активную форму
Save Project As	Сохраняет активную форму под другим именем
Save All Close	Сохраняет файл проекта под другим именем
Close All Use	Сохраняет файл проекта и все открытые модули

Репозиторий Delphi

Окно репозитория Delphi открывается при выборе File | New |

На его страницах расположены пиктограммы для выбора прототипов форм, модулей, проектов и экспертов построения форм и проектов. Зависимые переключатели Copy, Inherit и Use определяют режим связи между хранящимся в репозитории прототипом и его копией в проекте: Copy - выбранный элемент копируется в текущий каталог и автоматически подключается к проекту; между образцом и его копией нет никакой связи; inherit - в проекте создаются наследники выбранного элемента и всех его родителей; любые изменения образца проявляются во всех проектах, которые его унаследовали; изменения наследника не влияют на образец; Use - выбранный элемент становится частью проекта; любые его изменения в проекте приводят к изменениям образца и сказываются во всех других проектах, которые его унаследовали или используют.

Опция Edit

Undo	Отменяет последнее изменение проекта
Redo	Восстанавливает последнее изменение проекта
Cut	Вырезает выбранный компонент формы или фрагмент текста и помещает его в буфер Clipboard

Copy	Копирует в Clipboard выделенные компоненты формы или фрагмент текста модуля
Paste	Извлекает из буфера и переносит компоненты на форму или копирует текст в модуль (в позицию, указываемую текущим положением текстового курсора)
Delete	Удаляет выделенные компоненты или фрагмент текста
Select All	Выделяет все компоненты формы или весь текст модуля
Align To Grid	Привязывает выделенные компоненты к масштабной сетке так, чтобы их левые верхние углы располагались в ближайших точках сетки
Bring To Front	Перемещает выделенные компоненты на передний план
Send To Back	Перемещает выделенные компоненты на задний план
Align	Вызывает окно выравнивания выделенных компонентов
Size	Вызывает окно изменения размеров выделенных компонентов
Scale	Масштабирует выделенные компоненты
Tab Order	Изменяет порядок обхода компонентов клавишей Tab
Creation Order	Изменяет порядок создания невизуальных компонентов
Flip Children Lock Controls Add to interface	Изменяет свойство BiDiMode для всех или только для выделенных компонентов Блокирует возможность перемещения компонентов на форме Определяет новые свойства, методы и события для компонентов ActiveX

Управление группой компонентов

Чтобы выделить группу компонентов, нужно нажать и удерживать клавишу Shift, после чего щелкать по компонентам левой кнопкой мыши. Каждый отмеченный таким образом компонент выделяется серыми квадратиками по углам его видимых границ. Затем вызываются опции Align, size или Scale для соответственно выравнивания, изменения размеров или масштабирования компонентов. Другой вариант - не отпуская Shift, щелкнуть правой кнопкой мыши по любому свободному месту формы и выбрать нужную опцию в локальном меню.

Переключатели этого окна определяют выравнивание всех выделенных компонентов относительно самого первого выделенного компонента (эталона):

- Left sides - компоненты выравниваются по левой границе;
- Centers - компоненты центрируются относительно границ эталона;
- Right sides-компоненты выравниваются по правой границе эталона;
- Space equally - между всеми компонентами устанавливается равное расстояние по горизонтали или вертикали;
- Center in window-все компоненты центрируются относительно границ окна;
- Tops - компоненты выравниваются по верхнему краю;
- Bottoms - компоненты выравниваются по нижнему краю.

Группа переключателей width регулирует ширину выделенных компонентов, а группа Height - их высоту:

- Shrink to smallest-уменьшает размеры компонентов до размеров наименьшего из-них;
- Grow to largest-увеличивает размеры компонентов до размеров наибольшего из них;
- Width - указывает ширину компонентов;
- Height - указывает высоту компонентов.

С помощью окна Scaling factor вводится коэффициент масштабирования в процентах от текущих размеров.

Опция Search

Find	Ищет фрагмент текста и подсвечивает его, если он найден
Find In Files	Ищет фрагмент текста во всех файлах проекта, или только в открытых файлах, или, наконец,
Replace	Ищет и заменяет фрагмент текста во всех файлах текущего каталога
Search Again	Повторяет поиск или поиск и замену
Incremental Search	Ищет текст по мере его ввода - сначала первую букву, затем две первые буквы и т. д.
Go to Line Number	Перемещает курсор на строку с указанным номером от начала файла
Browse Symbol	Показывает место возникновением определения символа программы (опция доступна только после успешного прогона программы). Символом считается любой глобальный идентификатор вашего проекта

Find Error	По адресу ошибки периода прогона программы отыскивает фрагмент кода

Опция View

Project Manager	Показывает окно Менеджера проекта
Translation Manager	Открывает доступ к Менеджеру трансляций Показывает окно Инспектора объектов
Object Inspector	Показывает окно
Object TreeView	Дерева объектов
To_Do List	Открывает доступ к списку To-Do
Alignment Palette	Показывает окно палитры выравнивания компонентов
Browser	Показывает окно браузера объектов
Code Explorer	Показывает спрятанное ранее окно Навигатора кода
Component List	Показывает окно для выбора компонентов
Window List	Показывает окно открытых окон проекта
Debug Windows	Отладочные окна. Эта опция-заголовок открывает доступ к подменю со следующими опциями
Breakpoints	Показывает окно точек останова
Call Stack	Показывает окно стека
Watching expressions	Показывает окно наблюдения за переменными/выражениями
Local Variables	Позволяет наблюдать за изменениями локальных переменных в отладочном режиме
Threads	Показывает окно статуса потоков команд
Modules	Показывает окно модулей проекта
Event Log	Показывает журнал событии
CPU	Показывает состояние регистров центрального процессора
FPU	Показывает состояние регистров арифметического сопроцессора
Desktops	Управляет конфигурациями основных окон. Эта опция-заголовок открывает доступ к подменю со следующими опциями

Save Desktop	Сохраняет текущую конфигурацию
Delete	Удаляет ранее сохраненную конфигурацию
Save Debug Desktop	Определяет текущую конфигурацию как отладочную
Toggle Form/Unit	Переключает активность из окна формы в окно кода программы и обратно
Units	Показывает окно модулей
Forms	Показывает окно форм
Type Library	Показывает окно библиотеки типов
New Edit Window	Открывает новое окно с кодом текущего модуля
Toolbars	Показывает окно настроек панелей инструментальных кнопок
View as Form/Text	Показывает окно формы в обычном виде или в виде текстового описания
Next Window	Показывает следующий присоединенный к проекту модуль

Менеджер проекта.

Центральную часть окна менеджера проекта занимает список всех связанных с проектом форм. Кнопка New открывает доступ к репозиторию, чтобы добавить новый модуль к проекту. Кнопка Remove удаляет модуль из проекта. После щелчка правой кнопкой мыши по модулю появляется локальное меню, с помощью которого можно открыть модуль или сохранить его под другим именем.

С помощью менеджера проектов осуществляется компиляция проекта на тот или иной язык локализации. Если для проекта определены языки локализации, в группе проектов помимо исполняемого файла xxxx.exe будут дополнительные проекты с тем же именем, но разными расширениями (по одному проекту на каждый язык локализации). Если, например, язык локализации английский (Великобритания), в группе будет проект xxxx.eng. Сделайте активным нужный проект и вызовите опцию Project | Build xxxx (xxxx - имя вашего проекта). В результате будет откомпилирован нужный ресурсный файл. После, этого вновь активизируйте проект xxxx.exe, с помощью опции proj-ect | Language | Set Active укажите язык локализации и вновь откомпилируйте проект. К нему будет подключен нужный ресурсный файл, и ваша программа станет локализованной.

Менеджер трансляции

Менеджер трансляций упрощает создание локализованных версий программных продуктов. Он становится доступным только после указания языков, на которые будут переводится текстовые сообщения, надписи, оп-

ции и другие текстовые ресурсы программы. Для каждого языка создается своя динамически подключаемая библиотека ресурсов. Изменение этой библиотеки перед компиляцией программы изменяет ее язык (см. выше “Менеджер проекта”).
Для выбора языка (языков) локализации используется опция меню Project /Language. Главным языком программы по умолчанию считается язык локализации Windows, так что если Delphi работает под управлением русскоязычной Windows, главным языком будет русский.
Список TO-DO
Список to-do предназначен для координации работы нескольких программистов в рамках одного проекта.
Этот список содержит все комментарии проекта, которые начинаются символами *//todo:*
Комментарии содержат сообщения руководителя проекта и/или программистов об обнаруженных ошибках и обычно располагаются там, где обнаружена неточность.
После вызова списка то-do переход к нужному комментарию осуществляется двойным щелчком мыщи на соответствующей строке списка.
Программист может сообщить об устранении ошибки, отметив переключатель в левой части строки списка, - в этом случае текст строки выводится перечеркнутым шрифтом.
В списке можно указать приоритет сообщения (колонка “! ”), его собственника (Owner) и категорию.
Эти параметры можно задать с помощью соответствующих ключей непосредственно в комментарии или после вызова редактора сообщения в списке (он вызывается после активизации сообщения и нажатия клавиши F2).
Браузер объектов
Браузер объектов доступен только после успешного прогона программы. Он представляет в наглядной форме используемые в проекте и доступные объекты, позволяя просмотреть их иерархию и входящие в них свойства и методы.
Опция Project

Lanages	Позволяет добавить новый, удалить или сделать главным один из существующих языков локализации
View Source	Показывает окно с кодом проекта Опция-заголовок.
Add To Repository	Помещает проект в репозиторий
Import Type Library	Импортирует в проект библиотеку типов элементов ActiveX
Remove From Project	Удаляет файл из проекта

Add To Project	Добавляет файл к проекту
Add Remove Set Active Update Resources DLL	Добавляет новый язык локализации Удаляет существующий язык локализации Делает активным язык локализации Создает заново ресурсные DLL, управляющие локализацией программы
Add New Project	Добавляет программу, DLL или пакет к текущей проектной группе
Add Exists Project	Открывает проект и добавляет его к текущей проектной группе
Compile Project!	Компилирует модули, которые изменились с момента предыдущей компиляции проекта
Build Projecti	Компилирует все модули проекта и создает исполняемую программу
Syntax Check Project1	Проверяет синтаксическую правильность программы
Information	Показывает информацию о вашей программе
Compile All Projects	Компилирует все файлы данной проектной группы, которые изменились с момента предыдущей компиляции
Build All Projects	Компилирует все файлы данной проектной группы независимо от того, изменялись ли они или нет с момента последней компиляции

Web Deployment Options	Устанавливает ActiveX компонент или ActiveForm на вашем Web-сервере. Вызывается перед компиляцией проекта
Web Deploy	Устанавливает ActiveX компонент или ActiveForm на вашем Web-сервере. Вызывается после компиляцией проекта
Options	Показывает диалоговое окно установки параметров проекта

Выбор языков локализации

После выбора Project /Languages I Add вызывается окно эксперта выбора языков локализации проекта.

В нем указывается один или несколько проектов, для которых осуществляется локализация, и главный язык проекта (код $419 соответствует русскому языку). В следующем окне (после щелчка по кнопке Next) вы сможете

выбрать языки локализации, отметив их флажками. На следующем шаге эксперт предложит вам указать каталоги размещения библиотек, которые будут содержать локализованные ресурсы. Все библиотеки имеют одинаковое название xxxx. drc (xxxx - имя проекта), поэтому они должны размещаться в разных каталогах. Для компиляции программы с тем или иным языком локализации используются менеджер проекта и опция Project | Language I Set Active.

Управление опциями проекта

Управление опциями проекта осуществляется с помощью диалогового окна, вызываемого опцией Project | Options.

На странице Forms окна опций проекта указывается главная форма проекта, а также автоматически создаваемые формы (Auto-create forms) и доступные проекту формы (Available forms). Кнопки между панелями этого окна позволяют переносить формы из одной панели в другую. Переключатель Default разрешает/запрещает использовать текущие установки страницы как умалчиваемые для других проектов.

На странице Application указывается подпись под пиктограммой свернутой программы (Title), сама пиктограмма (Icon) и имя Help-файла (Help file).

На странице Compiler собраны переключатели, управляющие параметрами процесса компиляции. В том числе (в фигурных скобках указана соответствующая директива компилятора): Optimizations - включает режим оптимизации *{$0},* Aligned record fields - размещает данные, выравнивая их на границу 32-разрядного слова *{$a};* Stack frames - заставляет компилятор создавать стековые рамы для всех процедур и функций *{$W}*. Pentium-Safe fdiv - вырабатывает код, предохраняющий от ошибок в вещественных вычислениях на процессорах Pentium ранних выпусков *{$U};* Range Checking - создает код проверки выхода за границы массивов *{$R};* l/o Checking создает код проверки корректности выполнения операций ввода/вывода *{$I};* Overflow checking - вырабатывает код проверки переполнения при выполнении целочисленных операций *($Q}*. Strict var-Strings - определяет строгую проверку соответствия строковых типов при обращении к подпрограммам *{$V}*. Complete Boolean Eva1 - определяет полное вычисление логических выражений *{$B};* Extended Syntax - включает расширенный синтаксис Object Pascal *{$xf;* Typed @ Operator - контролирует типы указателей в операции @ *{$T};* open Parameters - разрешает использование открытых параметров в подпрограммах *{$p}*. Huge Strings - связывает зарезервированное слово String с длинными строками *{$H};* Assignable Typed Constants - разрешает присваивание типизированным константам *{$J};* Debug information - помещает в DCU-файл отладочную информацию *{$D};* Local Symbols - создает отладочную информацию о локальных символах программы *{$l};*

Symbol information - создает отладочную информацию о символах программы *{$y},* Show Hints - заставляет компилятор выдавать рекомендации; Show Warnings - заставляет компилятор выдавать предупреждающие сообщения; Assertions - заставляет вырабатывать код для отладочных процедур Assertion *{$C}.*

Страница Linker определяет параметры компоновщика, в том числе: Off - запрещает создавать карту распределения памяти; Segments - карта содержит список модулей и адреса точек входа всех подпрограмм; Publics - дополняет Segments отсортированным списком символов секций public; Detailed - дополняет Public детальной информацией о модулях; Generate dcus - создает стандартные для Delphi DCU-фаЙлы; Generate C Object files - создает файлы в формате объектов языка C; Generate C++ Object files - создает файлы для связывания с программой, созданной С помощью C++ Builder; Generate Console Application - создает консольную программу; include TD32 Debug info - помещает в исполняемый файл информацию для внешнего отладчика; include remote debug symbols-используется для удаленного вызова отладчика; Min Stack Size-устанавливает минимальный размер стека; мax Stack size - устанавливает максимальный размер стека; image Base - указывает начальный адрес для загрузки изображений (для dll); exe Descriptor - информационная строка длиной до 255 символов, которая включается в исполняемый файл (например, для объявления авторских прав).

Страница Directories/Conditionals задает каталоги размещения и условные символы: Output Directory - указывает каталог размещения исполняемого файла; Unit Output Directory - указывает каталог размещения Dcu-файлов;

Search Path - каталог размещения файловых с исходными текстами программы, если они не обнаружены в текущем каталоге; при необходимости указать несколько каталогов в любом из описанных выше окон они разделяются точкой с запятой; Debug source path - определяет каталог размещения внешнего отладчика; bpl output directory - указывает папку размещения файлов компиляции пакетов; dcp output directory - определяет каталог размещения файлов для локализации программ; Conditional Defines - определяет символы для условной компиляции; Unit Aliases-определяет псевдонимы модулей.

На странице Versioninfo сосредоточены параметры управления информацией о версии программы: include version information in project - если переключатель выбран, в проект включается информация о версии программы, которую можно прочитать после щелчка правой кнопкой мыши на пиктограмме программы и выборе Properties; Module Version Number - поля Major, Minor, Release, Build определяют составной номер версии; Auto-increment build number - если переключатель активен, номер версии автоматически наращивается при каждой компиляции программы; Debug Build

- указывает на создание отладочной версии программы; Pre-Release - указывает на создание некоммерческой версии программы; Special Build - казывает на специальную версию программы; Private Build - указывает на версию, не предназначенную для широкого распространения; dll - создается динамическая библиотека; Language id - идентификатор языка, на который рассчитана программа.

Опция Run

Run	Компилирует программу и делает ее прогон
Attach to Process	Позволяет присоединиться в режиме отладки к одному из уже запущенных процессов на другой сетевой машине
Parameters	Указывает командную строку запуска вашей программы
Register ActiveX Servers	Регистрирует ваш проект в реестре Windows. Опция доступна для ActiveX-проектов
Unregister ActiveX Servers	Удаляет ваш проект из реестра Windows. Опция доступна для ActiveX-проектов
Install MTS Objects	Регистрирует в вашем проекте объект MTS
Step Over	В отладочном режиме выполняет текущую строку кода и не прослеживает работу вызываемых подпрограмм
Trace Into	В отладочном режиме выполняет текущую строку кода и прослеживает работу вызываемых подпрограмм
Trace To Next Source Line	Программа выполняется до ближайшего от текущего положения курсора исполняемого оператора
Run To Cursor	В отладочном режиме выполняет программу и останавливается перед выполнением кода в строке с текстовым курсором
Run Until Return	В отладочном режиме выполняет текущую подпрограмму и останавливается
Show Execution Point	Отображает в окне кода оператор, на котором было прервано выполнение программы
Program Pause	Приостанавливает прогон отлаживаемой программы

Program Reset	Прекращает прогон программы и восстанавливает ре жим конструирования программы
Inspect	Открывает окно проверки текущего значения
Evaluate/Modify	Открывает окно проверки/изменения переменных
Add Watch	Добавляет переменную или выражение в окно наблюдения
Add Breakpoint	Добавляет точку останова

Опция Component

New Component	Открывает окно эксперта компонентов
Install Component	Помещает компонент в существующий или новый пакет
Import ActiveX Control	Добавляет к проекту библиотеку типов ActiveX-компонентов
Create Component Template	Помещает шаблон в палитру компонентов
Install Packages	Указывает пакеты, необходимые на этапе конструирования и прогона программы
Configure Palette	Вызывает диалоговое окно настройки палитры компонентов

Опция Database

Explore	Вызывает инструмент исследования баз данных - Database Explorer или SQL Explorer (в зависимости от версии Delphi)
SQL Monitor	Вызывает инструмент запросов к БД - SQL Monitor
Form Wizard	Вызывает окно эксперта форм для создания формы, отображающей наборы данных из удаленных или локальных БД

Опция Tools

Environment Options	Вызывает окно настройки параметров среды Delphi и ее инструментов

Editor Options	Вызывает окно настройки параметров редактора Delphi
Debugger Options	Вызывает окно настройки параметров отладчика Delphi

Настройка параметров среды

Диалоговое окно настройки параметров среды вызывается опцией
Tools I Environment Options**.**

Закладка Preferences открывает доступ к параметрам среды Delphi: Editor files - перед прогоном автоматически сохраняются все измененные файлы; Project Desktop - перед прогоном автоматически сохраняется информация о состоянии экрана; Desktop Only - при выходе из программы сохраняется информация о состоянии экрана, Desktop and Symbols - при выходе из программы сохраняется информация о состоянии экрана и символах программы на момент последней удачной компиляции; Auto drag docking - разрешает причаливать одно инструментальное окно к другому; Show compiler progress - показывать окно отображения процесса компиляции; Warn on package rebuild - предупреждать о перекомпиляции пакетов; Minimize On Run - минимизировать окна Delphi в момент старта программы; Hide Designers On Run - прятать вспомогательные окна (окно Инспектора объектов и окна форм) в момент старта программы; Directory -содержит путь к окну расположения файла репозитория DELPHI32.DRO; если путь не указан, используется каталог bin каталога размещения Delphi.

Закладка Designer содержит настройки для процесса конструирования форм:

Display Grid - показывать сетку на пустой форме; Snap to Grid - привязывать расположение компонентов к узлам сетки; Show component captions - показывать имена компонентов на этапе конструирования программы; show designer hints - показывать оперативную подсказку об именах компонентов и их типах на этапе конструирования; show extended control hints - показывать ярлычки оперативной подсказки с расширенной информацией; New Forms as Text - сохранять файлы описания форм в текстовом формате; Auto create forms & data modules - при выборе переключателя каждая добавляемая к проекту форма или модуль данных помещается в список автоматически создаваемых форм, в противном случае - в список доступных форм; Grid size x - горизонтальный шаг сетки; Grid size Y - вертикальный шаг сетки.

Закладка Object Inspector открывает окно настройки параметров Инспектора объектов: SpeedSettings - позволяет выбрать один из следующих вариантов настройки: Custom colors & settings (настраиваемые цвета и установки). Default colors & settings (умалчиваемые цвета и установки), Delphi 5 emulation (эмуляция Delphi 5), Visual Studio (TM) emulation (эмуляция

Visual Studio); Colors - позволяет настроить цветовыделение опций Инспектора объектов; Expand inline - раскрывать свойства и методы внешнего объекта; Show on events page -показывать внешний объект на странице событий.

Закладка Palette предоставляет средства для настройки палитры компонентов: вы может изменять порядок следования компонентов и страниц, переименовывать страницы, добавлять к ним новые компоненты, удалять существующие и т. д.

Закладка Library определяет каталоги размещения библиотек Delphi: Library path - маршрут поиска библиотечный файлов; bpl output library - выходной каталог для размещения BPL-файлов; dcp output library - выходной каталог для размещения DCP-файлов; Browsing Path - каталоги для, браузера.

Закладка Explorer позволяет настроить свойства кодового браузера, окно которого обычно “причалено” к окну редактора кода. Automatically show Explorer - если этот переключатель отмечен, окно кодового браузера появляется при создании каждого нового проекта; Highlight incomplete class items - выделяет цветом незавершенные определения классов; show declaration syntax - позволяет помимо имен элементов показывать свойства и методы; Explorer sorting- определяет способ сортировки элементов (по алфавиту или по порядку объявления). Finish incomplete properties - после отметки завершает определения не только классов, но и свойств. Переключатели initial browser view позволяют выбрать приоритет показа классов, модулей или глобальных определений.

Закладка Type Library служит для управления свойствами редактора библиотеки типов (используется при разработке многозвенных приложений баз данных).

Закладка Environment variables определяет т. н. переменные среды разработки, в том числе тип ОС, положение основных файлов ОС и т. п.

С помощью небольшой странички Delphi Direct можно управлять автоматической связью с сайтом Delphi для обновления информации об этом программном продукте.

Закладка internet управляет форматами создаваемых файлов для передачи по Интернет (интранет).

Закладка General окна кодового редактора (вызывается Tools I Editor Options) позволяет настроить общие свойства кодового редактора: Auto indent Mode- реализовать автоотступ (при нажатии Enter курсор устанавливается на начало предыдущей строки); insert Mode - определяет умалчиваемым режим вставки (переключается клавишей Insert); Use Tab Character - при нажатии клавиши Tab в текст вставляется символ Tab (если переключатель не установлен, вставляются символы пробела); Smart Tab - при нажатии клавиши Tab смещает курсор к первому не пробелу в предыдущей

строке; Optimal Fill - начинает каждый автоотступ с минимального количества символов Tab и/или пробела; Backspace Unindents - разрешает удалять автоотступ клавишей Backspace; Cursor Through Tabs - разрешает курсору перескакивать через пустые символы табуляции; Group Undo - разрешает удалять группу последних изменений текста при нажатии Alt+Backspace или выборе опции Edit/Undo; Cursor Beyond eof - позиция курсора включает символы конца строки; undo After Save - восстанавливать изменения, бывшие до последнего сохранения файла; Keep Trailing Blanks - сохранять ведомые символы пробела; Brief Regular Expressions - использовать шаблоны при поиске и поиске-замене; шаблоны включают в себя следующие специальные символы:

Символ	**Назначение**
^ или %	В начале строки означает, что искомый образец должен располагаться с начала строки
& или >	В конце строки означает, что искомый образец должен располагаться в конце строки
?	На этом месте может стоять любой символ
@	После символа указывает на любое число этих символов в этом месте: bo@ означает boo, bo, bot
+	После символа указывает на любое число символов, которые следуют дальше: bo+ означает boo, bonus, bot, но не b или bo
|	Выбор одного из выражений - до или после черты: bar | car выбирает bar или car На этом месте не должен стоять следующий символ
~	На этом месте должен стоять любой из указанных в скобках символов: [bot] означает b, o или t
[^]	На этом месте не должен стоять любой из указанных в скобках символов: [bot] означает любой символ, кроме b, o или t
[-]	Задает диапазон символов. Например [bot] -любой символ в диапазоне от b до o включительно
( )	Определяет вложенный шаблон. Среда допускает до 10 уровней вложенности
\	Отменяет действие стоящего за ним специального символа

Persistent Blocks - указывает, что выделенный блок остается выделенным, даже если его покидает текстовый курсор (выделение сохраняется до нового выделения);
Overwrite Blocks - заменяет выделенный блок текстом из буфера; если при этом установлен переключатель persistent Blocks, блок вставляется сразу за

выделенным; Double Click Line - выделяет цветом всю строку при двойном щелчке на любом ее символе; Find Text At cursor - если переключатель установлен, в образце поиска при поиске или поиске/замене будет появляться слово, рядом с которым располагается текстовый курсор; Force Cut And Copy Enabled - разрешает операции Edit/Cut и Edit/Copy, даже если текст не выделен; Use syntax Highlighting - разрешает использование синтаксического цветовыделения; Block indent - определяет величину автоотступа для выделенного блока; undo Limit - определят размер буфера для операций Undo; Tab stops - размер табулостопа в символах шрифта экрана; Syntax Extensions - определяет расширения файлов, для которых будет использоваться синтаксическое цветовыделение.

Закладка Display определяет настройку экрана: Brief Cursor Shapes - использовать формы курсора, подобные использующимся в редакторе Brief; Create Backup File - создавать страховочный файл с расширением, начинающимся символом “~” (тильда); Preserve Line Ends - исключать остановку текстового курсора в конце строки; Zoom To Full Screen - разрешить распахивание кодового окна на весь экран; visible Right Margin - разрешает показывать правую границу текста в виде вертикальной пунктирной линии; visible gutter - разрешает показывать служебную зону в левой части окна редактора; Right Margin - определяет правую границу текста; Gutter width - определяет ширину левой служебной зоны; Editor Font - определяет используемый в редакторе шрифт; size - задает высоту шрифта.

Закладка Color позволяет выбрать цвета для отдельных синтаксических элементов. Эти цвета будут использованы для синтаксического цветовыделения.

Закладка Key Mappings позволяет настроить “горячие” клавиши, используемые в кодовом редакторе для ускорения решения типовых задач.

Закладка Code insight определяет используемые интеллектуальные возможности кодового редактора: Code Completion - разрешает использовать подсказку в виде списка свойств, методов и событий, появляющуюся после ввода имени класса (или имени объекта) и следующей за ним точки; Code Parameters - разрешает появление подсказки с перечислением формальных параметров при определении вызовов методов класса; Tooltip Expression Evaluation - разрешает показывать подсказку, в которой на этапе останова программы в контрольной точке будет отображаться текущее содержимое переменной или выражения при указании на них мышью; Tooltip Symbol insight - разрешает показывать информацию об объявленном типе и местонахождении этого объявления; Code Completion Delay - определяет задержку в секундах до включения интеллектуальных возможностей редактора; Templates - определяет кодовые слова, по которым редактор подготавливает стандартные заготовки; для использования уже определенных кодовых слов просто напечатайте нужное слово и нажмите Ctrl+Shift+J;

например, если вы напечатаете arrayd и нажмете Ctrl+Shift+j, редактор вставит array **[0..I] of ;**
(вертикальная черта обозначает место появления текстового курсора);
кнопки Add, Edit и Delete используются соответственно для добавления нового, редактирования и удаления старого кодового слова; Code - показывает код, который связан с указанным кодовым словом.
Опция Help

Delphi Help	Основная справочная служба Delphi
Delphi Tools	Справочная служба по инструментам Delphi
Windows SDK	Справочная служба по Windows API
Borland Home Page	Домашняя страничка Borland
Delphi Home Page	Домашняя страничка Delphi
Borland Developer Support	Страничка поддержки разработчиков
Delphi Direct	Окно Интернет-поддержки разработчика
Customize	Вызов службы OpenHelp
About	Окно с краткой информацией о Delphi

Служба OpenHelp
Служба OpenHelp предназначена для модификации справочной службы Delphi: с ее помощью можно удалять ненужные разделы и, что наиболее важно, вставлять новые. Последнее связано с тем, что существует множество относительно небольших фирм, занимающихся разработкой и продажей пакетов компонентов для Delphi и C++ Builder (пакеты компонентов для этих двух инструментов идентичны), а также компонентов ActiveX. Сама Delphi имеет развитые средства создания новых. Поставщики компонентов обычно поставляют вместе с ними нужные файлы справочной помощи, которые OpenHelp может сделать частью справочной службы Delphi.
Служба OpenHelp вызывается опциями Help | Customize главного меню.
Четыре закладки этого окна управляют содержанием справочной службы (закладка Content), набором файлов помощи (index), контекстно-чувствительной помощью (Link) и проектами (Project). Чтобы удалить раздел содержания, файл справки или проект, нужно на соответствующей закладке щелкнуть по удаляемому компоненту и выбрать в меню Edit | Remove Files или щелкнуть по инструментальной кнопке.
Для добавления к содержанию нового раздела требуется предварительно создать два специальных файла: файл с таблицей содержания (с расширением .toe) и файл содержания (.cnt). toe-файл, как следует из документации, есть стандартный файл содержания cnt, создаваемый утилитой hcw.exe за тем отличием, что в нем нельзя использовать секции include.

Файл cnt, в свою очередь, отличается от стандартного тем, что в нем должны быть только две директивы: base и title, причем содержимое title у обоих файлов должно быть одинаковым и они должны располагаться в одной папке. После создания файлов вызывается OpenHelp, и на закладке Content щелкается кнопка или вызывается опция Edit |Add Files. В появляющемся вслед за этим диалоговом окне нужно указать положение файла toe.
Для добавления справочного файла (.hlp) выбирается закладка index, щелкается та же кнопка и указывается положение hlp-файла.

8.2 Работа с редактором

Все команды редактора можно разделить на команды перемещения курсора, команды удаления/вставки, команды работы с блоками, прочие.
При их описании используются следующие обозначения клавиш управления курсором:

вл курсор влево; вв курсор вверх;
вп курсор вправо; вн курсор вниз.

Команды перемещения курсора

вл	на символ влево	HOME	в начало строки
вп	на символ вправо	END	в конец строки
Ctrl-ВЛ	на слово влево	Ctrl+PgUp	в начало экрана
Ctrl-ВП	на слово вправо	Ctrl+PgDn	в конец экрана
вв	на строку вверх	Ctrl+HOME	в начало файла
вн	на строку вниз	Ctrl+END	в конец файла
PgUp	на страницу вверх	Ctrl+Q+B	в начало блока
PgDn	на страницу вниз	Ctrl+Q+K	в конец блока

Команды удаления/вставки

INS	включить/отключить режим вставки	Delete	стереть символ справа от курсора
Enter	вставить строку	Ctrl+T	стереть слово справа от курсора
Qrl+Y	удалить строку	Ctrl+Q+Y	стереть остаток строки справа от курсора
Backspace	стереть символ слева от курсора	Ctrl+Z	отменить последнее изменение текста

Команды работы с блоками
При подготовке текстов программ часто возникает необходимость перенести фрагмент текста в другое место или удалить его. Для такого рода опе-

раций удобно использовать блоки - фрагменты текста, рассматриваемые как единое целое. Длина блока может быть достаточно большой, он может занимать несколько экранных страниц. В каждый момент в одном окне редактора может быть объявлен только один блок. Обмен блоками между окнами возможен только через буфер Clipboard.

Ctrl+K+T	пометить в качестве блока слово слева от курсора
Ctrl+K+P	напечатать блок
Ctrl+K+H	убрать выделение блока цветом; повторное использование Ctrl+K+H вновь выделит блок
Ctrl+K+Y	удалить блок
Ctrl+K+R	читать блок из дискового файла в позицию, определяемую текстовым курсором
Ctrl+K+W	записать блок на диск
Ctrl+K+I	сместить блок вправо на два символа
Ctrl+K+U	сместить блок влево на два символа
Shift+Delete	вырезать блок и поместить его в Clipboard
Ctrl+Insert	копировать блок в буфер Clipboard
Shift+Insert	вставить содержимое Clipboard в позицию, указываемую текстовым курсором

Прочие команды

Ctrl+F F3	искать по образцу продолжить поиск
Ctrl+R	искать по образцу и заменить
Ctrl+K+n	установить маркер; *n = 0..9* (см. ниже)
Ctrl+Q+n	искать маркер
Ctrl+Q+]	искать парную скобку (см. ниже)
Ctrl+O+O	вставить настройку компилятора в начало файла (см. ниже)
Ctri+Shift+R	начинает и заканчивает определение макроса
Ctrl+Shift+P	выполняет ранее определенный макрос

Ctrl+K+n. Устанавливает в текущую позицию курсора маркер с номером *n = 0..9.* Маркер на экране появляется в виде небольшого окошка с номером маркера в служебном поле слева от текста. Он никак не влияет на исполнение программы. Команда используется совместно с командой **Ctrl+Q+n** (искать маркер с номером и) для ускорения поиска нужных фрагментов текста при разработке крупных программ. Раз установленный маркер нельзя удалить, но можно его поместить в другое место файла. При

записи на диск маркеры не запоминаются, т. е. после чтения файла с диска в нем нет маркеров.
Ctrl+Q+]. Эта команда используется для поиска ближайшей парной скобки. Она позволяет отыскивать пары скобок (и), { и }, [и]. Подведите курсор так, чтобы он расположился непосредственно перед одной из скобок, и дайте команду - редактор отыщет нужную парную скобку.
Ctrl+0+0. Эта команда заставит редактор поместить в самое начало файла строки, содержащие текущую настройку среды в виде директив компилятора.
В окне кода можно запрограммировать часто повторяющиеся манипуляции с клавишами в виде макроса. Для начала указания макроса используется команда **Ctri+Shift+R**. После этого любые действия программиста с клавиатурой запоминаются вплоть до повторения команды **Ctrl+Shift+R.** Запомненный макрос исполняется командой **Ctrl+Shift+P.**
Поиск объявлений
Если активизировать окно кода и перемещать в нем указатель мыши при нажатой и удерживаемой клавише Ctrl, текст программы приобретает свойства гипертекста: на идентификаторах стандартный указатель мыши заменяется на руку с пальцем, а соответствующий идентификатор выделяется цветом и подчеркиваеся. Если в этот момент нажать левую кнопку мыши, редактор попытается отыскать исходный текст модуля, в котором объявлен соответствующий тип, подпрограмма или глобальная переменная, и, если поиск окажется удачным, загрузит текст модуля в окно кода и установит в нем текстовый курсор в начале описания типа (подпрограммы, переменной). Такого же эффекта можно достичь, если щелкнуть по идентификатору правой кнопкой мыши и выбрать опцию Find Declaration.
Поиск идет в следующем порядке (ниже указаны опции меню и элементы соответствующих окон, содержащие нужные каталоги):

1. Project | Options | Directories | Conditionals | Search path;
2. Project | Options|Directories | ConditionalsI Debug source path;
3. Tools | Environment Options[Library|Browsing path
4. Tools | Environment Options | Library | Library path.

Нельзя искать объявления, если текущий программный модуль еще ни разу не был сохранен на диске.
Если мышь перемещается над идентификатором без нажатой и удерживаемой клавиши Shift, рядом с указателем мыши появляется небольшое окно, в котором сообщается, к какому элементу языка относится идентификатор (к процедуре, функции, переменной и т. д.), а также имя модуля и номер строки в нем, где этот идентификатор впервые описан.
Создание стандартных заготовок для новых свойств и методов
При объявлении новых свойств класса в интерфейсной секции вы можете написать лишь имя свойства и его тип.

После нажатия Ctrl+Shift+C или щелчка правой кнопкой мыши и выбора продолжения Complete Class at Cursor редактор добавит необходимые элементы Read и Write в описание свойства и внесет другие изменения в текст программы.
Пусть, например, вы написали

```
type TMyButton = class(TButton) property Size: Integer;
procedure DoSomething;
end;
```

и нажали Ctrl+Shift+C (текстовый курсор при этом должен находиться в любом месте внутри описания класса). Редактор изменит описание класса следующим образом:

```
type TMyButton = class(TButton)
property Size:
Integer read FSize write SetSize;
procedure DoSomething;
private
FSize: Integer;
procedure SetSize(const Value: Integers;
end;
```

и добавит в исполняемую секцию описание двух методов:

```
{ TMyButton }
procedure TMyButton.DoSomething;
begin
end;
procedure TMyButton.SetSize(const Value: Integer);
begin
FSize := Value;
end;
```

Вы можете также вставить в раздел implementation новый метод класса и нажать Ctrl+Shift+C - редактор вставит прототип метода в объявление класса в секции interface.

Навигация внутри модуля

Клавиши курсора *Вверх* (Up) и *Вниз* (Down) в сочетании с нажатыми и удерживаемыми клавишами Ctrl и Shift осуществляют переключение между секциями interface и implementation текущего модуля. Если в секции interface вас заинтересовала реализация того или иного метода, щелкните по нему мышью и нажмите Ctrl+Shift+Down - редактор отыщет реализацию и покажет ее вам. Наоборот, чтобы из секции implementation переместиться к заголовку метода в секции interface, нажмите Ctrl+Shift+Up.

Вставка текстовых заготовок и окно Code Insight

Редактор может вставлять в текст множество текстовых заготовок, позволяющих сэкономить время ввода кода программы.

Просмотреть имеющиеся текстовые заготовки и при необходимости добавить к ним собственные можно с помощью Tools | Editor Options | Code Insight.
В средней части окна с помощью списка Templates можно выбрать имя и краткое описание образца, а в окне Code увидеть и при желании отредактировать соответствующую текстовую заготовку.
С помощью кнопки Add можно добавить новый образец, с помощью Edit — изменить его имя и краткое описание, а с помощью Delete - удалить его.
При редактировании имеющейся или вставки новой заготовки учтите, что символ “|” определяет позицию текстового курсора после вставки заготовки в текст программы.
Для вставки заготовки напечатайте ее имя в окне кода и нажмите Ctrl+J - имя заменится на полный текст заготовки.
Если вы не помните всех имен заготовок, их можно выбрать из списка.
Для этого установите текстовый курсор в то место, где вы хотите вставить заготовку, и нажмите Ctrl+J - на экране появится окно с именами и краткими описаниями всех заготовок.
Остальные элементы окна Code insight:
Code Completion - разрешает/запрещает появление окна с именами свойств и методов класса после ввода имени объекта и точки (рис. П 1.15).
Code Parameters - разрешает/запрещает появление окна с именами формальных параметров обращения к подпрограмме после ввода имени подпрограммы и открывающей скобки (рис. П 1.15).
Tooltip Expression Evaluations - разрешает/запрещает появление окна с указанием текущего значения переменной, над именем которой располагается указатель мыши в режиме отладки
Tooltip Symbol Insight - разрешает/запрещает появление окна с указанием параметров идентификатора, над которым располагается указатель мыши.
Delay - определяет задержку в секундах включения механизма показа окон Code Insight.

8.3 Отладка программ

В Delphi имеется мощный встроенный отладчик, значительно упрощающий отладку программ. Основными инструментами отладки являются точки контрольного останова и окно наблюдения за переменными.
Точки контрольного останова
Точка контрольного останова определяет оператор в программе, перед выполнением которого программа прервет свою работу и управление будет передано среде Delphi. Точка останова задается с помощью опции view | Debug windows | Breakponts.

Окно точек останова содержит список всех установленных в проекте точек, перед выполнением которых происходит прекращение работы программы и управление получает среда Delphi.
Для добавления новой точки следует щелкнуть по окну правой кнопкой мыши и выбрать опцию Add. В этом случае появляется окно, с помощью которого можно указать положение добавляемой точки: FileName - определяет имя файла;
Line number - номер строки от начала файла (в момент появления окна оно содержит файл и строку с текстовым курсором). В строке Condition можно указать условие останова в виде логического выражения (например, MyValue = Max-Value-12), а в строке Pass count - количество проходов программы через контрольную точку без прерывания вычислений.
Окно наблюдения
Наблюдать за состоянием переменной или выражения можно с помощью специального окна, вызываемого опцией View | Debug windows | Watches
Окно наблюдения используется в отладочном режиме для наблюдения за изменением значений выражений, помещенных в это окно. Для добавления нового выражения щелкните по окну правой кнопкой мыши и выберите опцию New Watch. В строке Expression введите выражение. Окно Repeat count определяет количество показываемых элементов массивов данных; окно Digits указывает количество значащих цифр для отображения вещественных данных;
переключатель Enabled разрешает или запрещает вычисление выражения. Остальные элементы определяют вид представления значения. Замечу, что в последних версиях Delphi вы можете просмотреть в отладочном режиме текущее значение любой переменной, если укажите на нее курсором: значение появится в ярлычке рядом с курсором.
Принудительное прерывание работы программы
Если программа запущена из среды Delphi, ее работу можно прервать в любой
момент G помощью клавиш Ctrl+F2, кнопки, опцией Run | program pause или, наконец, установив точку контрольного останова в той части программы, которая выполняется в данный момент или будет выполнена.
Трассировка программы
Перед исполнением оператора, в котором установлена точка контрольного останова, работа программы будет прервана, управление получит среда Delphi, а в окне наблюдения отразится текущее значение наблюдаемых переменных и/или выражений. Теперь программист может прослеживать работу программы по шагам с помощью клавиш F7 и F8 или инструментальных кнопок. При нажатии F8 будут выполнены запрограммированные в текущей строке действия, и работа программы прервется перед выполнением следующей строки текста программы. Замечу, что контрольная точка

останова выделяется по умолчанию красным цветом, а текущая прослеживаемая строка - синим. Если программа остановлена в контрольной точке, т.е. когда текущая строка совпадает со строкой останова, строка выделяется красным цветом, Признаком текущей строки является особое выделение строки в служебной зоне слева в окне редактора.

Кстати, чтобы установить/снять точку контрольного останова, достаточно щелкнуть мышью по служебной зоне слева от нужной строки или установить в эту строку текстовый курсор и нажать F5.

При нажатии F7 среда выполняет те же действия, что и при нажатии F8, однако, если в текущей строке содержится вызов подпрограммы пользователя, программа прервет свою работу перед выполнением первого исполняемого оператора этой подпрограммы, т.е. клавиша F7 позволяет прослеживать работу вызываемых подпрограмм.

После трассировки нужного фрагмента программы можно продолжить нормальную ее работу, нажав клавишу F9.

Действия в точках прерывания

В Delphi 5 и 6 с любой точкой можно связать одно или несколько действий. Для этого нужно активизировать окно точек останова, вызвать его локальное меню (щелчок правой кнопкой) и выбрать продолжение Properties. В появившемся окне свойств щелкнуть по кнопке Advanced.

В нижней части окна имеется панель Actions, с помощью которой и определяются действия для точки останова, указанной в верхней части окна.

Break - простой останов перед выполнением помеченного оператора.

ignore subsequent exceptions - если переключатель установлен, игнорируются все возможные последующие исключения в текущем отладочном сеансе до очередной точки останова, в которой, возможно, это действие будет отменено.

Handle subsequent exceptions - после установки этого переключателя отменяется действие предыдущего переключателя и возобновляется обработка возможных исключений.

С помощью Log message вы можете указать произвольное сообщение, связанное с точкой останова, а с помощью Eval expression - вычислить некоторое выражение и поместить его результат в это сообщение.

Группировка точек прерывания

В Delphi 5 и 6 имеется возможность объединения точек останова в группы. Для этого используется все то же окно рис. П1.20: в строке Group следует указать имя группы, к которой принадлежит точка, а в строках Enable Group и Disable Group соответственно разрешить или запретить действие всех точек останова, относящихся к соответствующей группе.

Вычисление выражений и изменение значений

С помощью окна Evaluate/Modify можно узнать значение любого выражения или установить в переменную другое значение. Это окно вызывается в режиме отладки после нажатия Ctrl+F7.
Это окно - модальное, т. е. оно прерывает отладку программы до тех пор, пока не будет закрыто. В строке Expression можно написать имя переменной или интересующее вас выражение. После щелчка по кнопке Evaluate в поле Result появится текущее значение переменной (выражения). Если в Expression содержится имя переменной, одновременно становится доступной кнопка Modify, а в строке New value повторяется текущее значение переменной. Если изменить эту строку и нажать Modify, в переменную будет помещено новое значение, которое и будет использоваться при дальнейшем прогоне программы (если определяется значение выражения, кнопка Modify и строка New value будут недоступны).
В Delphi 5 и 6 используются также дополнительные кнопки этого окна Watch и inspect. Если вы щелкните по первой из них, выражение (переменная) из окна Evaluate будет перенесено в окно наблюдений watch, щелчок по второй отображает выражение (переменную) в специальном окне Inspect.

Ведение протокола работы программы

В ряде случаев бывает неудобно или невозможно пользоваться пошаговой отладкой программ. Если вы, например, установите точку останова в подпрограмме прорисовки сетки TDBGrid, программа после останова не сможет нормально продолжить свою работу, т. к. в этом случае она будет пытаться восстановить экран и вновь будет остановлена и т. д. В таких ситуациях вам могут помочь контрольные точки, которые не прерывают работу программы, а лишь помещают некоторую информацию в специальный файл трассировки. Для реализации такой точки раскройте окно RunlAdd Breakpoint | Source Breakpoint, уберите флажок в переключателе Break и напишите сообщение в строке Log message. Вы можете также в строке Eval expression указать некоторое выражение, которое будет вычислено и вместе с сообщением помещено в протокол работы программы. Этот протокол можно просмотреть в любой момент (в том числе и после завершения прогона программы) с помощью опции View] Debug Windows! Event Log.

Библиография

1. А. Епанешников, В. Епанешников. Программирование в среде Turbo Pascal 7.0. – 3-е изд., стер. – М.: «ДИАЛОГ-МИФИ», 1997. – 288с.
2. Фаронов В.В. Турбо Паскаль 7.0. Начальный курс. Учебное пособие. – М.: «Нолидж», 1997. – 616с., ил.
3. Сидоров М. Е. , Трушин О. В. Школа работы на IBM PC. Часть 2. Уфа, 1996.
4. Григорьев С.А. Программирование на языке Паскаль для математиков: Учебное пособие / Калинингр. ун-т. - Калининград, 1997. - 92 с.
5. М.Б. Львовский. Методическое пособие по информатике для учащихся 9-11 классов, изучающих IBM PC.
6. Шауцукова Л.З. Информатика 10 - 11. — М.: Просвещение, 2000 г.
7. Практикум по информатике: Учеб. пособие для студ. высш. учеб. заведений / А.В. Могилев, Н.И. Пак, Е.К. Хеннер; Под ред. Е.К. Хеннера. – М.: Издательский центр «Академия», 2002. – 608с.
8. Культин Н.Б. Программирование в Turbo Pascal 7.0 и Delphi. 1999г.
9. Культин Н.Б. Turbo Pascal в задачах и примерах. 2000г.
10. Немнюгин С.А. Turbo Pascal: практикум. 2001г.
11. А.Б. Николаев, Л.А. Акатнова "Турбо-Паскаль в примерах". Книга для учащихся 10-11 кл. М.: 2002. - 111с.
12. И.Г. Семакин, А.П. Шестаков "Основы программирования". Учебник для среднего образования М: 2003. - 617с.
13. Климова Л.М. "Турбо-Паскаль". Учебное пособие для абитуриентов. М.: 2002. - 497с.
14. Практикум по программированию на языке Паскаль (Массивы, строки, файлы, рекурсия, указатели) Издание четвертое. Автор: М. Э. Абрамян. Ростов-на-Дону: ООО «ЦВВР», 2004. - 187 с.: ил. (http://sunschool.math.rsu.ru/books/pascal1.htm#start)
15. Д.А. Сурков, К.А. Сурков, А.Н. Вальвачев. Программирование в среде Borland Pascal для Windows: Справ. пособие. – Минск: Выш.шк., 1996. – 432с.: ил.
16. Очков В.Ф., Пухначев Ю.В. 128 советов начинающему программисту. – 2-е изд. – М.: Энергоатомиздат, 1992. – 256с.: ил.
17. В. Кучеренко. Тонкости программирования на Delphi. Серия книг «Кратко, доступно, просто» - М.: «Познавательная книга плюс», 2000. – 192с.
18. Бобровский С.И. Delphi 5: начальный курс. М.: «Десс», «Информком-Пресс», 1999. – 272с.
19. Калверт Ч. Delphi 4. Самоучитель: Пер. с англ./ Ч. Калверт. – Киев: Изд-во «ДиаСофт», 1999. – 192с.

20. Александровский А.Д., Шубин В.В. Delphi для профессионалов. Опыт практического применения. – М.: ДМК, 2000. – 240с.: ил. (Серия «Для программистов»).
21. Гаевский А.Разработка программных приложений на Delphi 6 – М.: Киев, 2000.
22. Джон Матчо, Дэвид Р.Фолкнер. «Delphi» - пер. с англ. – М.:Бином, 1998.
23. Коцюбинский А.О., Грошев С.В. Язык программирования Delphi 5 – М.: «Издательство Триумф», 1999.
24. Леонтьев В. Delphi 5 – М.: Москва «Олма-Пресс», 1999.
25. Моисеев А. Object Pascal – М.: Москва, 2000.
26. Ремизов Н. Delphi – М.: Питер, 2000.
27. Справочная система Delphi 5.0 Help.
28. Т.А. Ильина. Программирование на Delphi 6 – М.: Питер, 2000.
29. Фаронов В. В. Delphi 4. Учебный курс. М.: Нолидж, 1999.
30. Федоров А. Г. Создание Windows-приложений в среде Delphi. М.: ТОО «Компьютер Пресс», 1999.
31. Хендерсон К. Руководство разработчика баз данных в Delphi 2. Киев: Диалектика, 1998.
32. Шапошников И. Delphi 5 – М.: Санкт-Петербург, 2001.
33. Шумаков П. В. Delphi 3 и разработка приложений М.: Нолидж, 1999.