Министерство образования Российской Федерации

АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра общей математики и информатики

Алгоритмизация и программирование. Основы программирования в Delphi 5.

Учебно–методическое пособие

Благовещенск 2001

Учебно-методическое пособие «**Алгоритмизация и программирование. Система программирования Delphi 5**» по курсу «Информатика» / Т.А. Макарчук - Благовещенск: Амурский гос. ун-т. 2001.

Учебно-методическое пособие включает в себя теоретические сведения по алгоритмизации и программированию. В пособии даются начальные сведения по системе программирования Delphi 5, разработанной американской корпорацией Borland International Inc. и языку программирования Object Pascal. Пособие рассчитано как на начинающих программистов, так и имеющих опыт программирования. Умение пользоваться операционной системой Windows 95/98/2000 является непременным условием усвоения пособия.

Рецензенты:

С О Д Е Р Ж А Н И Е

ВВЕДЕНИЕ. 5

ЧАСТЬ 1. АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ.

ТЕМА 1. АЛГОРИТМИЗАЦИЯ. 6

1.1 Понятие алгоритма, его свойства, способы представления. 6

1.2. Основные базовые структуры алгоритма. 7

ТЕМА 2. ЯЗЫКИ И СИСТЕМЫ ПРОГРАММИРОВАНИЯ. 10

2.1. Уровни языков программирования. 10

2.2. Интерпретаторы и компиляторы. 11

2.3. Поколения языков программирования. 11

2.4. Обзор языков программирования высокого уровня. 12

2.5. Системы программирования. 14

ТЕМА 3. ПРОГРАММИРОВАНИЕ. ПРОЕКТИРОВАНИЕ ПРОГРАММ. 16

3.1. Структурное программирование. 16

3.2. Модульное программирование. 17

3.3. Объектно-ориентированное программирование. 18

3.4. Этапы проектирования и создания программ. 19

ЧАСТЬ 2. СИСТЕМА ПРОГРАММИРОВАНИЯ DELPHI.

ТЕМА 4. ЗНАКОМСТВО СО СРЕДОЙ DELPHI. 21

4.1. Delphi. Основные характеристики продукта. 21

4.2. Структура среды программирования. 22

ТЕМА 5. ОСНОВЫ ВИЗУАЛЬНОГО ПРОГРАММИРОВАНИЯ. 26

5.1. Delphi - объектно-ориентированная среда. 26

5.2. Конструирование окна формы. Свойства. 27

5.3. События. Реакция на события. Окно кода программы. 29

5.4. Сохранение, компилирование, запуск программы. 30

ТЕМА 6. ОБЗОР КОМПОНЕНТ. 31

6.1 Компоненты страницы Standard. 31

6.2 Компоненты страницы Additional. 32

6.3. Некоторые компоненты дополнительных страниц. 33

ЧАСТЬ 3. ЯЗЫК ПРОГРАММИРОВАНИЯ OBJECT PASCAL.

ТЕМА 7. СТРУКТУРА ПРОГРАММ. 34

7.1. Структура проекта. 34

7.2. Структура модуля. 35

7.3. Структура подпрограмм. 37

ТЕМА 8. ЭЛЕМЕНТЫ ЯЗЫКА. 39

8.1. Основные понятия языка. 39

8.2.Данные. Структура типов данных. Простые типы данных. 40

8.3. Математические выражения, функции и операции. 43

8.4. Операции отношения. Логические операции. 44

ТЕМА 9. ОПЕРАТОРЫ ЯЗЫКА. 46

9.1. Оператор присваивания. 46

9.2. Составной оператор. 46

9.2. Составной оператор. 46

9.3. Условный оператор. 47

9.4. Оператор выбора. 49

9.5. Циклические операторы. 50

ТЕМА 10. СТРУКТУРИРОВАННЫЕ ДАННЫЕ. 52

10.1 Массивы. 52

10.2. Строки. 55
10.3. Записи. 57
10.4. Множество. 58
ТЕМА 11. ФАЙЛЫ. 60
11.1. Общие принципы работы с файлами. 61
11.2.Текстовые файлы. 60
11.3.Типизированные файлы. 63
11.4.Нетипизированные файлы. 63
Часть 4. КОМПОНЕНТЫ
ТЕМА 12. ОБЩИЕ ХАРАКТЕРИСИКИ КОМПОНЕНТОВ. 66
12.1. Иерархия компонентов. Родительские и дочерние компоненты. 66
12.2. Общие свойства компонентов 68
12.3. Общие методы компонентов. 70
12.4. Класс TStrings – набор строк. 73
12.5. Многооконное приложение. Диалоговые окна. 74
ТЕМА 13. ИСПОЛЬЗОВАНИЕ КОМПОНЕНТОВ ОБЩЕГО НАЗНАЧЕНИЯ. 80
13.1. TFrame – рама. 80
13.2. TLabel – метка. 80
13.3. Tedit – однострочное окно ввода – вывода. 81
13.4. TMemo - многострочное окно ввода – вывода. 82
13.5. TButton – командная кнопка. 83
13.17. TDrawGrid – произвольная таблица. 84
13.7. TCheckBox – флажок. 84
13.8. TRadioButton – радиокнопка. 86
13.9. TRradioGroup – панель радиокнопок. 86
13.10. TGroupBox – панель группирования. 87
13.11. TPanel – панель. 87
13.12. TListBox – список выбора. 88
13.13. TComboBox - раскрывающийся список выбора. 89
13.14. TMainmenu – главное меню. 90
13.15. TPopupMenu – контекстное меню. 91
13.16. TSringGrid - таблица строк. 92
13.17. TDrawGrid – произвольная таблица. 95
13.8. TBitBtn – командная кнопка с изображением. 95
13.19. TUpDown – спаренная кнопка. 96
13.20. TOleContainer – контейнер объектов OLE. 97
ТЕМА 14. ГРАФИЧЕСКИЙ ИНСТРУМЕНТАРИЙ. 99
14.1. Класс TFont. 99
14.2. Классы TPen, TBrush. 99
14.3. Класс TCanvas. 100
14.4. Компонент Tshape, TImage, TPaintBox, TChart. 103
ТЕМА 15. КОМПОНЕНТЫ СТРАНИЦА DIALOGS. 107
15.1. Компонент TColorDialog - выбор цвета. 108
15.2. Компонент TFontDialog – диалог выбора шрифта. 108
15.3. Компоненты TOpenDialog и TSaveDialog – диалоги открытия и сохранения файлов. 109
15.4. Компоненты TOpenPictureDialog и TSavePictureDialog - диалоги открытия и сохранения изображений. 111
ЛИТЕРАТУРА. 112

ВВЕДЕНИЕ

Это учебно-методическое пособие рассчитано на студентов, которые самостоятельно или под руководством преподавателя пытаются научиться основам программирования.

Лучшим языком для изучения программирования является Pascal с системой программирования TurboPascal для MS-DOS. После появления графической операционной системы Windows 3.1, разработчики систем программирования, прежде всего корпорации Wicrosoft и Borland, не замедлили выпустить соответствующие средства: в 1991г. Borland выпускает TurboPascal for Windows, в 1992г. - усовершенствованную версию этой системы программирования - Borland Pascal with Object 7.0. В 1993 г. вышла первая визуальная среда программирования Visual Basic фирмы Wicrosoft, в 1995г. корпорация Borland выпустила первую версию Delphi, а затем, с интервалом в один гг - ещё четыре версии: 2, 3, 4 и 5.

Система Delphi является продолжением языка Паскаль, поддерживающей методику объектно-ориентированного программирования и создающей приложения для Windows. Программирование в Delphi стало проще, чем программирование для MS-DOS в Pascal.

В первой части этого пособия рассматриваются основные понятия алгоритмизации и программирования. Последующие главы посвящены изучению основ работы в Delphi 5. Во второй части описывается среда Delphi: основные характеристики продукта, структура среды программирования, основы визуального программирования в среде. Третья часть посвящена изучению языка Object Pascal, без детального знания которого невозможна работа с Delphi. В четвертой части рассматриваются компоненты общего назначения, формы и программа в целом. В пособии не рассматриваются компоненты для работы с базами данных, Интернет и ряд других не так часто используемых, но весьма полезных компонентов.

ЧАСТЬ 1. АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ

Тема 1. Алгоритмизация

1.1. ***Понятие алгоритма, его свойства, способы представления.***

Алгоритм — понятное и точное предписание к исполнителю совершить последовательность действий, направленных на решение задач определенного типа.

Свойства алгоритма:

дискретность (алгоритм должен быть разбит на конкретные действия, выполнив один шаг, приступает к исполнению следующего);

понятность (точное понятие команды);

определенность (исполнитель алгоритма не должен сомневаться в следующем шаге);

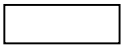
результативность (по завершении выполнения алгоритма должен быть получен результат);

массивность (алгоритм должен быть написан для решения ряда подобных задач).

Способы представления алгоритма: словесный, математическая формула, табличный, графический (блок-схема).

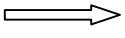
Для записи блок-схемы используются следующие обозначения:

 - начало (конец) алгоритма

 - команды (действия)

 - ввод (вывод) данных

 - условие

 - направление, указание следующего действия дейст-

1.2. ***Основные базовые структуры алгоритма.***

Основными базовыми структурами алгоритма являются:

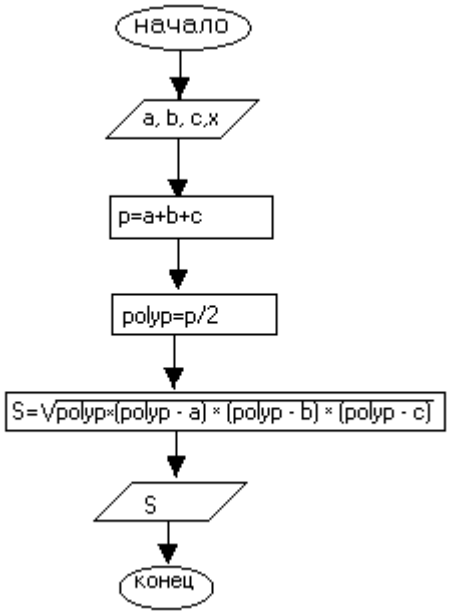
линейная структура;

разветвляющая структура (ветвление);

циклическая структура (цикл).

Линейная структура – алгоритмическая структура, в которой команды выполняются последовательно.

Примером линейной алгоритмической структуры может служить построение алгоритма нахождения площади треугольника по сторонам треугольника a,b,c. Площадь вычисляется с использованием формулы Герона.

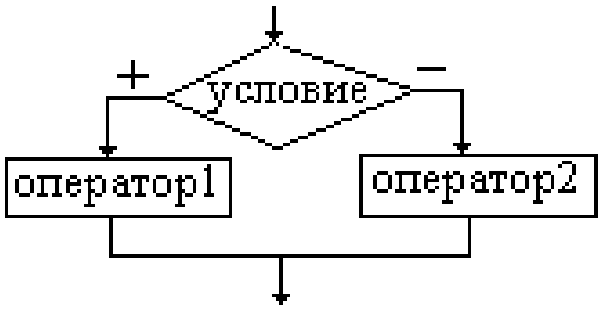


Разветвляющая структура (ветвление) – это алгоритмическая структура, в которой проверяется некоторое условие и в зависимости от результатов проверки выполняется то или иное действие.

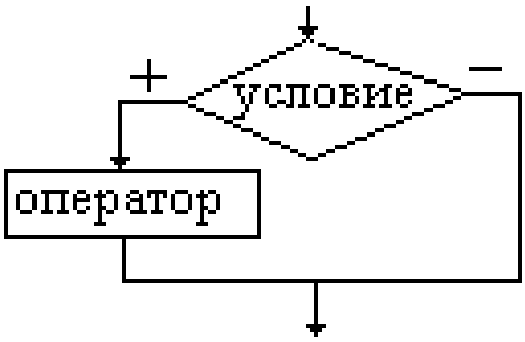
Существует два типа ветвления: *полное* и *неполное*.

Алгоритм полного ветвления работает по следующему принципу. Вначале вычисляется условие. Если результат условия *True* (истина), то выполняется оператор1, а оператор2 пропускается. Если результат условия False (ложь), то выполняется оператор2, а оператор1 пропускается.

Блок-схема полного ветвления имеет следующий вид:



Блок-схема неполного ветвления имеет следующий вид:

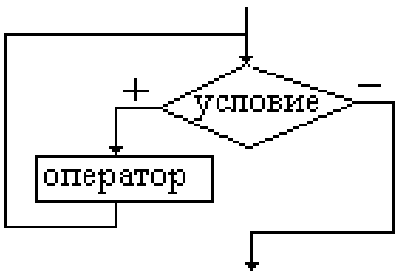


Неполная разветвляющая структура работает по следующему алгоритму. Сначала вычисляется условие. Если условие принимает значение TRUE, то выполняется оператор, иначе никаких действий не происходит.

Циклическая структура (цикл) – это алгоритмическая структура, содержащая повторяющиеся команды.

Циклические структуры бывают трех видов: счетный цикл (цикл с параметром), цикл с предусловием, цикл с постусловием.

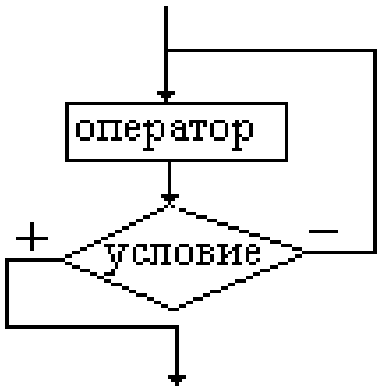
Блок-схема цикла с предусловием следующая:



Циклическая структура с предусловием работает по следующему алгоритму. Первоначально проверяется условие. Если оно истинно, то выполняется

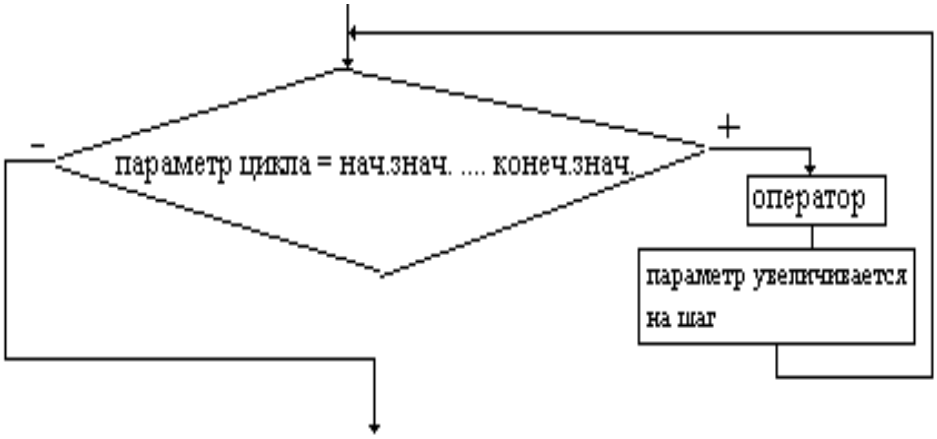
тело цикла (оператор). Затем снова проверяется условие и т.д. Если условие ложно, то цикл завершается и выполняется оператор, стоящий непосредственно после цикла.

Блок-схема цикла с постусловием следующая:



Циклическая структура цикла с постусловием работает по следующему алгоритму: сначала выполняется тело цикла (операторы), потом проверяется условие. Если условие ложно, то снова выполняются операторы, и так до тех пор, пока условие не примет значение *TRUE*.

Блок-схема счетного цикла (с параметром) следующая:



Циклическая структура счетного цикла работает по следующему алгоритму: параметру цикла присваивается начальное значение и выполняется тело цикла, затем параметр цикла увеличивается на шаг и вновь выполняется тело цикла, и так до тех пор, пока не будут перебраны все значения параметра цикла.

Тема 2. **Языки и системы программирования**

Язык программирования - формальный язык для описания алгоритма решения задачи на компьютере.

2.1. ***Уровни языков программирования.***

Языки программирования бывают двух уровней: низкого и высокого уровня.

Языки программирования низкого уровня близки к машинному коду и ориентированы на конкретные команды процессора.

Языком самого низкого уровня является *язык ассемблера*, который просто представляет каждую команду машинного кода, но не в виде чисел, а с помощью символьных условных обозначений, называемых *мнемоникой*. Каждой модели процессора соответствует свой язык ассемблера.

С помощью языков низкого уровня создаются очень эффективные и компактные программы, так как разработчик получает доступ ко всем возможностям процессора. С другой стороны, для этого требуется очень хорошо понимать устройство компьютера (отладка больших приложений затруднена, а результирующая программа не может быть перенесена на компьютер с другим типом процессора). Подобные языки обычно применяются для написания небольших системных приложений, драйверов устройств, когда важнейшими требованиями становятся компактность, быстродействие и возможность прямого доступа к аппаратным ресурсам.

Языки программирования высокого уровня значительно ближе и понятнее человеку, для описания алгоритма используется привычная для него форма. Особенности конкретных компьютерных архитектур (аппаратных средств) в них не учитываются, поэтому создаваемые тексты программ легко переносимы на другие платформы, имеющие программу перевода данного языка в машинный код.

2.2. ***Интерпретаторы и компиляторы.***

С помощью языка программирования создается не готовая программа, а только ее текст, описывающий ранее разработанный алгоритм. Чтобы получить работающую программу, надо этот текст перевести в машинный код. Для этого используются программы-трансляторы, которые бывают двух видов: компиляторы и интерпретаторы.

Программа-компилятор переводит исходный текст программы в машинный код и записывает его на диске в виде отдельного бинарного файла (информация в машинном коде). При запуске программы исполняется бинарный файл. Почти все системы программирования работают с компилятором.

Программа–интерпретатор работает совместно с исходным текстом. Каждая команда интерпретируется в машинный код и немедленно исполняется. Файл на машинном языке не создается. Программа, написанная в системе программирования, включающая интерпретатор, работает медленно. Поэтому интерпретаторы не нашли широкого применения.

В настоящее время с интерпретатором работают в основном языки программирования для Интернета. Например, с интерпретатором работает Java, Perl.

2.3. ***Поколения языков программирования.***

1-е поколение составляют языки, созданные в начале 50-х ггов, когда первые компьютеры только появились на свет. Это был первый язык ассемблера, созданный по принципу «одна инструкция – одна строка».

2-е поколение составляют языки программирования конца 50-х – начала 60-х гг. Тогда появился символический ассемблер, в котором существовало понятие переменой. Он стал полноправным языком программирования.

3-е поколение языков программирования относится к 60-м гг. В это время родились универсальные языки высокого уровня, с помощью которых появилась возможность решать задачи из любых областей. Такие качества языков

программирования высокого уровня как относительная простота, независимость от конкретного компьютера и возможность использования алгоритмических конструкций позволили резко повысить производительность труда программистов. Подавляющее большинство языков программирования этого поколения успешно применяется и сеггня.

4-е поколение языков программирования предназначено для реализации крупных проектов, повышена их надежность и скорость создания. 4-е поколение началось в 70-х гг. и продолжается до настоящего времени. Как правило, языки 4-го поколения имеют мощные функции (операторы, команды), для реализации которых на языках поколения младшего уровня потребовалось бы тысячи строк.

5-е поколение языков программирования появилось в середине 90-х гг. К ним относятся системы автоматического создания прикладных программ с помощью визуальных средств разработки, без знания программирования. Инструкции вводятся в компьютер в максимально наглядном виде с помощью методов, наиболее удобных для человека, не знакомого с программированием.

*2.4. **Обзор языков программирования высокого уровня.***

Fortran (Фортран) – это компилируемый язык, созданный в 50-е гг. Этот язык появился первым после языка ассемблера, поэтому удобство создания программы не так хорошо реализовано, как возможность получения эффективного машинного кода. В 2000 г. выпущен Фортран F2k.

Cobol (Кобол) – компилируемый язык, разработанный в начале 60-х гг. для применения в экономической области и решения бизнес-задач. Отличается большой «многословностью» **—** его операторы иногда выглядят как обычные английские фразы. Очень распространен среди программистов США.

Algol (Алгол) – компилируемый язык, созданный в 1960 г. Он был призван заменить Фортран, но из-за более сложной структуры не получил широкого распространения.

Pascal (Паскаль) – создан в конце 70-х гг. Во многом напоминает Алгол, но в нем ужесточены требования к структуре программы, за счет чего структура стала более наглядной и простой. Паскаль удобен для получения азов программирования, но также успешно применяется при создании крупных проектов.

Basic (Бейсик) – создавался в 60-х гг. в качестве учебного языка и очень прост в изучении. Для него имеются интерпретаторы и компиляторы.

C (Си) – создан в лаборатории Bell и первоначально планировался для замены ассемблера. Имеет возможность создавать эффективные и компактные программы, в то же время не зависеть от конкретного типа процессора. Си во многом похож на Паскаль и имеет дополнительные средства для прямой работы с памятью. В 1980 г. создано объектно–ориентированное расширение языка Си – **C^{++}($Си^{++}$)**.

Java (Джава, Ява) – язык, созданный компанией Sun в начале 90-х гг. на основе $Си^{++}$. Главная особенность этого языка – компиляция не в машинный код, а в платформенно-независимый байт-код (каждая команда занимает 1 байт). Этот байт-код выполняется с помощью интерпретатора – визуальной Java–машины, версии которой существуют сегодня для любой платформы. Благодаря наличию множества Java–машин, программы легко переносятся на уровне двоичного байт-кода. Сегодня язык Ява очень популярен. Пока основной его недостаток – невысокое быстродействие, так как язык Ява интерпретируемый.

Языки программирования баз данных – язык структурированных запросов SQL. Основан на мощной математической теории и позволяет выполнять эффективную работу, манипулируя не отдельными, а группами записей. Для управления большими базами данных используются СУБД. Практически в каждой СУБД, помимо поддержки языка SQL, имеется свой уникальный язык, ориентированный на особенности этой СУБД: Microsoft, IBM, Oracle, Software.

Языки программирования для Интернет отличаются характерными особенностями: являются интерпретируемыми, интерпретаторы для них распространяются бесплатно, а сами программы в исходных текстах. Такие языки называются *скрипт-языками*. Примерами языков программирования для Интернет являются HTML, Perl, Tcl/Tk, VRML.

Логические языки программирования используются в области создания искусственного интеллекта. Наиболее распространены Пролог, Ада.

2.5. ***Системы программирования.***

Для написания программы на языке программирования необходимо иметь на компьютере установленную соответствующую систему программирования.

Системы программирования — хорошо интегрированная система, включающая как минимум:

специализированный текстовый редактор (для написания текста программы);

компилятор для перевода текста программы в машинный код (в редких случаях — интерпретатор);

библиотека функций (подключенные модули);

редактор связей для связывания модулей (файлов с исходными текстами) и стандартных функций, находящихся в библиотеках;

исполнимый код - законченную программу с расширением .COM или .EXE, которую можно запустить на любом компьютере, где установлена операционная система, для которой эта программа создавалась;

справочную систему;

отладчик, позволяющий анализировать работу программы во время ее выполнения по шагам.

В последние несколько лет в программировании (особенно в программировании для операционной системы Windows) наметился так называемый *визуальный подход*. Он облегчает создание графических приложений, в таких сис-

темах имеется множество стандартных элементов управления и контроль за их работой. Подобные системы программирования называются *средами быстрого проектирования* RAD-среды.

Наиболее популярны следующие визуальные среды быстрого проектирования программ для Windows:

Basic: Microsoft Visual Basic

Pascal: Borland Delphi

C++: Borland C++ Bulider

Java: Symantec Café.

Тема 3. Программирование. Проектирование программ

Программа – алгоритм, записанный на языке программирования.

Цель программирования: написание программы для решения задачи на ЭВМ.

К создаваемой программе предъявляются следующие требования: надежность, универсальность, простота эксплуатации, легкость модификации, большая скорость выполнения, минимум входной информации.

На первых этапах программирования, при написании средних по размеру приложений (несколько тысяч строк исходного теста), удовлетворяющих требования к написанию программы, использовался *метод структурного программирования*. Дальнейшим его развитием стало *модульное программирование*. В середине 80-х гг. в программирование возникло новое *объектно-ориентированное программирование*. Данное программирование позволило создавать приложения объемом в сотни тысяч строк.

3.1. ***Структурное программирование.***

Идея структурного программирования заключается в том, что структура программы должна отражать структуру решаемой задачи, чтобы алгоритм решения был виден из исходного текста. Структурное программирование включает два принципа:

1) использование ограниченного набора базовых структур (при написании любой программы используют только три базовые структуры: линейная, ветвление, циклы);

2) принцип пошаговой детализации – программа разбивается на множество мелких подпрограмм (подзадач), каждая из которых выполняет одно из действий, предусмотренных исходным заданием. Подпрограмма должна иметь до 50 команд – критический порог для быстрого понимания цели подпрограммы. Итоговая программа складывается путем комбинации подпрограмм.

Принцип пошаговой детализации позволяет вести проектирование и разработку приложений **сверху вниз (нисходящее проектирование**). Первоначально при анализе задачи выделяется несколько подпрограмм, решающих самые глобальные задачи (например, инициализация данных, главная часть, завершение), потом каждая из этих задач детализируется на более низком уровне, разбиваясь в свою очередь на небольшое число других подпрограмм, и так происходит до тех пор, пока вся задача не окажется детализованной. Степень детализации программы может быть различной, но иерархическая схема должна давать представление об общем алгоритме обработке данных.

Такой подход удобен тем, что позволяет человеку постоянно мыслить на предметном уровне, не опускаясь до конкретных команд. Небольшие подпрограммы намного легче отлаживать, что существенно повышает общую надежность всей программы. Очень важная характеристика подпрограмм – возможность их повторного использования.

В Pascal подпрограммы реализованы в виде процедур и функций.

*3.2. **Модульное программирование.***

При составлении крупных программных проектов (когда количество мелких подзадач очень велико) целесообразно небольшие подпрограммы общей направленности объединить в долее крупные структуры – модули. Например, созданный первый модуль будет включать подзадачи работы с массивами, второй содержать подзадачи, реализующие математические вычисления, третий — подзадачи реализации графического интерфейса программы.

Модуль – это самостоятельная часть программы, имеющая определенное назначение. Модуль состоит из логически взаимосвязанных совокупностей функциональных элементов — библиотеки подзадач.

3.3. ***Объектно-ориентированное программирование.***

С появлением идеи Windows возник событийно–ориентированный подход. Идеология системы Windows основана на событиях. Щелкнул человек на кнопке, выбрал пункт меню, нажал клавишу или кнопку мыши – в Windows генерируется подходящее событие, которое реализуется в окне активной программы. Событийный подход является развитием идей нисходящего проектирования, когда постепенно определяются и детализируются реакции программы на различные события.

События могут быть *пользовательскими*, возникшими в результате действий пользователя (например, ввод на клавиатуре или нажатие кнопки мыши), *системными*, возникшими в операционной системе (например, сообщения от таймера) и *программными*, генерируемые самой программой (например, обнаружена ошибка и ее следует обработать).

В основе объектно-ориентированного программирования лежит понятие объекта. Любой объект окружающего мира имеет свойства (характеристики объекта), методы (описывающие, что объект может делать) и события, на которые данный объект может реагировать. События приводят, как правило, к изменению свойств, применению методов.

В языках программирования *понятие объекта реализовано как совокупность свойств, методов, событий и реакций на события.* Рассмотрим описание объекта окна Windows. Объект окно Windows имеет:

Свойства – характеристики объекта, его параметры (окно характеризуется цветом, размером, заголовком окна типом окна и т.д.).

Методы – действия объекта (для окна методами являются перемещение, минимизация, закрытие, открытие, уменьшение, увеличение размеров и др.)

События – изменения в состоянии объекта: *пользовательские, системные, программные* (например, щелчок мыши на системных кнопках в заголовке окна, щелчок мыши на командной кнопке, подведение указателя мыши к рамке окна и др.).

Реакция на события — описания действий, которые необходимо совершить при данном событии: изменить свойства объекта, выполнить действия (методы).

Примерами объектов в Windows являются разные виды окон (в том числе и диалоговые), командные кнопки ("ОК", "ДА","ОТМЕНА" и т.п.), горизонтальное и контекстное меню и др.

Объекты объединяются в классы.

Класс – совокупность объектов, характеризующихся общностью применяемых методов и свойств объекта.

Экземпляром данного класса называется каждый конкретный объект, имеющий структуру этого класса.

3.4. Этапы проектирования и создания программ.

Формулируются и анализируются требования к проекту. Этот этап самый важный, так как неправильная постановка цели проекта заставляет выполнять ненужную работу. На данном этапе определяются входные данные (их количество должно быть минимально), тип этих данных, результирующие (выходные данные) и их приблизительная оценка. На основе требований по разным методикам определяется примерный объем проекта и его трудоемкость, рассчитываются будущие трудозатраты и определяется его стоимость (если речь идей о программном продукте).

Выбирается методология разработки программы. Используя метод структурного программирования, решаемую основную задачу разбивают на отдельные подзадачи, которые могут быть объединены в модули. При использовании RAD-средств проектируется интерфейс пользователя, определяются объекты управления программой, события и реакция на эти события. Выбираются события и определяются подзадачи, которые при этом событии будут решаться.

Алгоритмизация. Разработка алгоритмической структуры каждой подзадачи.

Программирование. Реализация конкретной подзадачи на языке программирования, возможно в виде реакции на событие.

Тестирование и отладка. Когда программа закончена (готова работоспособная *альфа-версия*), она тестируется. Тестирование – обнаружение ошибок. Исправляются синтаксические ошибки, ошибки программы. Задавая входные данные, полученный результат сравнивают с ожидаемым, в случае расхождения приступают к отладке. Анализируются, в частности, устойчивость работы программы при вводе недопустимых или критических значений, при отсутствии информации (неуказании данных), при неверных действиях. Когда число ошибок становиться минимальным, начинается *бета-тестирование*. К такому тестированию привлекаются максимально возможное число сотрудников, и программа уже начинает частично функционировать в рабочем режиме.

Исполняемая программа. Для программы необходимо подготовить сопровождение – справочную информацию о разработчике, назначение и правила работы с программой.

Контроль качества. На этапе тестирования и отладки нельзя дать стопроцентную гарантию качества программы. Чем крупнее проект, тем больше в нем ошибок. Предусматривается непрерывный, сквозной контроль качества за программой: вносятся изменения, конфигурируется новыми решаемыми подзадачами – выпускается новая обновленная версия. В ходе функционирования программы постоянно должны появляться новые версии (раз в 2-3 года) для поддержания спроса и удовлетворения потребностей на возникающие новые решаемые задачи в рамках программы.

ЧАСТЬ 2. СИСТЕМА ПРОГРАММИРОВАНИЯ DELPHI

Тема 4. Знакомство со средой Delphi

Delphi предназначен для профессионалов-разработчиков корпоративных информационных систем, однако часто Delphi используют с чисто прикладной целью, чтобы быстро решить задачи, не привлекая для этого программистов со стороны.

Для установки Delphi существуют следующие требования к аппаратным и программным средствам:

Windows 3.1 и выше.

27 Mb дискового пространства для минимальной конфигурации.

50 Mb дискового пространства для нормальной конфигурации.

процессор 80386, а лучше 80486.

8MB ОЗУ.

Небольшие программы, созданные на Delphi, будут работать на любом компьютере. Другими словами, они не требуют того ОЗУ или скорости процессора, что необходимо для среды Delphi.

4.1. ***Delphi. Основные характеристики продукта.***

Delphi – это комбинация нескольких важнейших технологий.

1. Высокопроизводительный компилятор в машинный код. Компилятор, встроенный в Delphi, обеспечивает высокую производительность. Этот компилятор в настоящее время самый быстрый в мире, его скорость компиляции составляет свыше 120 тыс. строк в минуту на компьютере 486DX33. Он предлагает легкость разработки и быстрое время проверки готового программного блока. Кроме того, Delphi обеспечивает быструю разработку без необходимости писать вставки на Си или код вручную (хотя и это возможно). В процессе построения приложения разработчик выбирает из палитры компонент готовые. Еще до компиляции он видит результаты своей работы – после подклю-

чения к источнику данных их можно видеть отображенными на форме, можно перемещаться по данным, представлять их в том или ином виде.

2. Объектно-ориентированная модель компонент. В стандартную поставку Delphi входят основные объекты, образующие удачно подобранную иерархию из 270 базовых классов. Это позволяет разработчикам строить приложения весьма быстро из заранее подготовленных объектов, а также дает им возможность создавать собственные объекты для среды Delphi. Никаких ограничений по типам объектов, которые могут создавать разработчики, не существует.

3. Визуальное построение приложений из программных прототипов. Среда Delphi включает полный набор визуальных инструментов для скоростной разработки приложений (RAD – rapid application development). VCL – библиотека визуальных компонент, включающая стандартные объекты построения пользовательского интерфейса, объекты управления данными, графические объекты, объекты мультимедиа, диалоги и объекты управления файлами, управление DDE и OLE. В Delphi визуальные компоненты пишутся на объектном Паскале. Визуальные компоненты Delphi получаются открытыми для надстройки и переписывания.

4. Масштабируемые средства для построения баз данных.

4.2. ***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, Word for Windows, Excel – это все MDI приложения. 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. Окна могут перемещаться по экрану, частично или полностью перекрывая друг друга.

После запуска Delphi экран монитора приобретёт вид, показанный на рис. 4.1 (на рисунке показаны окна Delphi 5, для других версий окна могут иметь незначительные различия):

1) главное окно (заголовок Delphi5 - Project1);
2) окно формы (заголовок Form1);
3) окно Инспектора Объектов (Object Inspector);
4) окно кода программы или окно редактора кода (заголовок Unit1.pas).

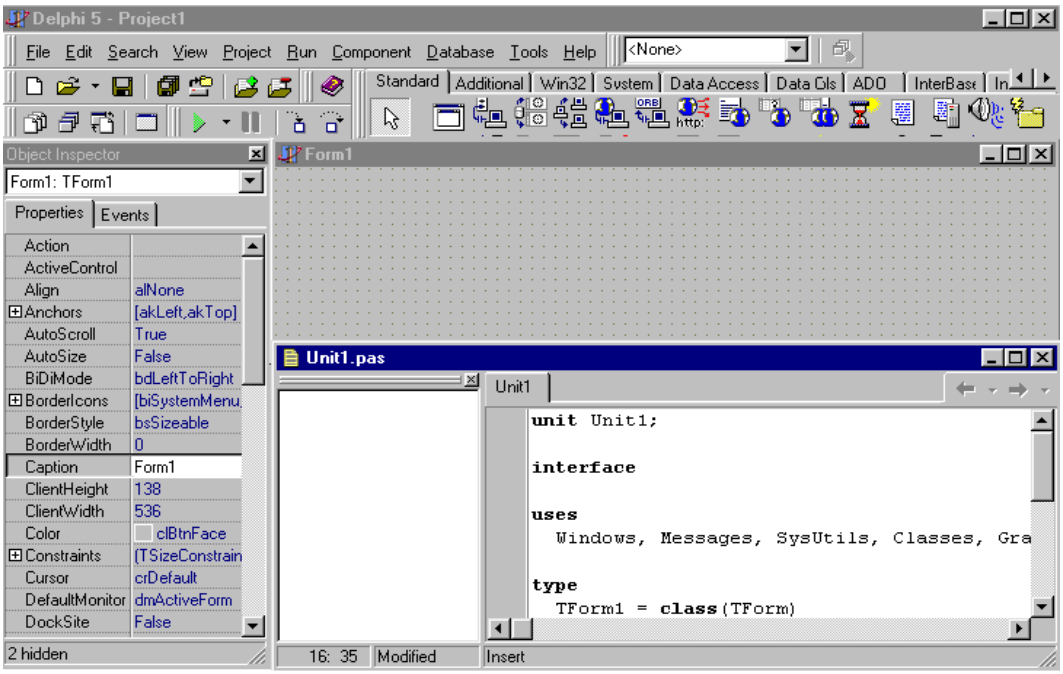


Рис. 4.1. Наиболее важные окна Delphi.

Чтобы упорядочить окна так, как они показаны на рисунке, их размеры и местоположение можно изменить вручную (как вы делаете это с окнами в Windows). Расположение окон не обязательно должно совпадать с рисунком, их расположение не влияет на работу Delphi. Первоначально окно кода перекрыто окном формы, переключение между которыми осуществляется клавишей **F12.**

Главное окно осуществляет основные функции управления проектом создаваемой программы. Это окно всегда присутствует на экране и занимает самую верхнюю часть. Закрытие окна означает окончание работы программиста с

системой программирования Delphi. В главном окне располагается главное меню Delphi (горизонтальная строка меню), панель инструментов SpeedBar и палитра компонент.

Панель инструментов SpeedBar находится непосредственно под меню, слева от Палитры Компонент. Пиктографические кнопки осуществляют быстрый доступ к наиболее важным командам горизонтального меню. Если задержать мышь над любой из кнопок на панели инструментов, то появится подсказка, объясняющая назначение данной кнопки. Назначение пиктографических кнопок показано в табл. 4.1.

Таблица 4.1. Назначение кнопок панели инструментов SpeedBar.

 - открытие файла (File/ Open file)
 - сохранение файла (File/ Save File)
 - добавляет новый файл к проекту (Projekt/Add to project)
 - удаляет файл из проекта (Project/Remove from project)
 - выбирает модуль из списка модулей, входящих в текущий проект (View/Units)
 - выбирает форму из списка форм, связанных с текущим проектом (View/Forms)
 - переключает между окном формы и окном кода (View/ Toggle Form/Units) - F12
 - создает новую форму и добавляет её к проекту (File/New Form)
 - компилирует и выполняет программу (Run/Run) - F9
 - реализует паузу в работе ослеживаемой программы (Run/Program Pause)
 - пошаговая трассировака программы с прослеживанием работы вызываемых подпрограмм (Run/Trace Into)
 - пошаговая трассировка программы, без прослеживания работы вызываемых подпрограмм (Run/Step Over)

Палитра компонент содержит визуальные компонентов VC. Компоненты сгруппированы в отдельные страницы, каждая из которых снабжена закладкой

(рис.4.2). Если вы щелкнете мышью на одну из закладок, то сможете перейти на соответствующую ей страницу. При подведении курсора мыши к компоненте появляется подсказка - название компоненты.

Рис. 4.2 Палитра компонент.

Форма – окно будущей программы. Окно формы представляет собой проект окна Windows: имеет заголовок, кнопку вызова системного меню, кнопку максимизации, минимизации и закрытия окна, рамку окна. На форме размещаются компоненты, тем самым формируя окно будущей программы.

Окно Инспектора Объектов позволяет изменять свойства объекта: цвет, размер, положение и др., указывать события для компонента – его поведенческую сторону: реагирование на щелчок мыши, на нажатие клавиши, как будет вести себя в момент появления на экране или в момент изменения размеров окна и т.п. Окно Инспектора Объектов содержит две страницы: *Properties (Свойства)* и *Events (События)*. Каждая страница представляет собой двухколончатую таблицу, левая колонка которой содержит название свойства или события, а правая – конкретное значение. В верхней части окна Инспектора Объектов располагается раскрывающийся список всех помещенных на форму компонентов. Если вы случайно закрыли окно Инспектора Объектов, то открыть его можно при нажатии на клавишу F11 или командой *View/ Object Inspector*.

Окно кода программы предназначено для создания и редактирования текста программы. Этот текст составляется по определенным правилам. Совокупность правил записи текста называется языком программирования. В системе программирования Delphi используется язык программирования Object Pascal.

Тема 5. **Основы визуального программирования.**

Программирование в Delphi строится на тесном взаимодействии двух процессов: процесса конструирования окна программы (визуальное проявление) и процесса написания кода программы. Для написания кода используется окно кода, для конструирования программы – остальные окна Delphi, и прежде всего – окно формы. Между содержимым окон формы и кода существует неразрывная связь, которая основана на объектно–ориентированной среде Delphi.

5.1. ***Delphi - объектно-ориентированная среда.***

Напомним, в основе объектно-ориентированного программирования лежит понятия объекта.

Объект – *нечто существенное и различимое, совокупность свойств, методов, событий и реакций на события.*

В Delphi реализованы почти все объекты Windows, имеется возможность создания собственных объектов. Каждый объект имеет:

Свойства – характеристики, параметры объекта, которые задаются в окне Инспектора Объектов во вкладке Свойства;

Методы – действия объекта (что объект может делать) – описание действий над данными;

События – изменения в окружающей объект обстановке, указываются в Инспекторе Объектов на странице События;

Реакция на события – описания действий, которые необходимо совершить при данном событии. Реакция на события описывается в окне кода программа в виде указании методов и изменении свойств объекта.

Многие объекты имеют идентичную структуру и различаются только значениями свойств. В таких случаях создается новый тип, основанный на единой структуре объекта, который называется *классом*, а каждый конкретный объект, имеющий структуру этого класса, называется *экземпляром данного класса*.

В состав Delphi входит несколько сотен классов, определяющих мощные возможности этой системы программирования. Каждый отдельный компонент является экземпляром определенного класса. Имя класса всегда начинается с буквы Т.

Все классы, реализованные в Delphi, представлены в виде визуальных компонентов VC, размещенных на палитре визуальных компонентов (смотри рис.4.2, гл. 4).

5.2. ***Конструирование окна формы. Свойства.***

Форма – окно будущей программы. На форме размещаются компоненты, тем самым, формируя окно будущей программы.

Форма является компонентом, поэтому работа с формой осуществляется так же, как с любым объектом Delphi. Форма относится к классу TForm. При открытии новой формы определяется новый класс TForm1, который порожден стандартным классом TForm. Данная форма – экземпляр класса TForm1 с именем Form1. В окне кода программы будет написано

```
Type TForm1 = class(TForm)
   Private { Private declarations }
   Public { Public declarations }
end;
var  Form1: TForm1;
```

Свойства формы, как и любого другого компонента, задаются в окне Инспектора Объектов на странице *Properties*. На этой странице указаны все свойства активного (выделенного) объекта. Рассмотрим первые несколько свойств компонент.

Свойство Name – имя компоненты, экземпляра класса. Определяет, как этот компонент будет называться в создаваемой программе. Имена создаются средой Delphi по такому принципу: сначала идет название компонента (Form), а за ним – порядковый номер размещенного на форме компонента (1), т.е. если

добавить еще одну форму, то она получит имя Form2, следующее – Form3 и т.д. Имя, заданное по умолчанию, можно изменить, но при этом желательно использовать только английские буквы и цифры. **Запомните**! Свойство Name задается первоначально и потом желательно его не изменять, это имя компоненты, с которой вы будете работать при написании программы.

Например, вы разместили на форме Form1 две компоненты класса TLabel. Свойство Name (имя) первой компоненты – экземпляра класса по умолчанию будет Label1, имя второй компоненты будет Label2. Вы можете работать с заданными именами, а можете заменить свойство Name первой компоненты, например, Name = A. Тогда, к первой компоненте вы будете обращаться с именем A. Для второй компоненты вы можете задать, например, Name = B. Тогда ко второй компоненте вы будете обращаться с именем B. В окне кода программы будет запись

```
Type  TForm1 = class(TForm)
  A: TLabel;
  B: TLabel;
 private { Private declarations }
 public  { Public declarations }
end;
var Form1: TForm1;
```

Свойство Font – шрифт позволяет выбрать параметры шрифта. Это сложное свойство, при свойстве Font стоит знак «+», при нажатии на который открывается подменю дополнительных параметров. Свойство Font можно задать также с помощью кнопки в правой части строки.

Свойство Caption – надпись создает надпись, в соответствии с установленным шрифтом (свойств Font).

Свойство Color – цвет определяет цвет объекта.

Свойство ICON – значок позволяет с помощью специального редактора выбрать подходящий значок для отображения в заголовке формы. При двой-

ном щелчке на данном свойстве открывается редактор Picture Editior. Кнопка Load открывает окно Load Picture для выбора файла со значком, который можно найти через Поиск. Часто файлы с расширением .ico находятся в C:\Programm files\borland\delphi\imeges\icon. Для подтверждения выбора значка необходимо нажать клавишу ОК.

Свойство Width – определяет ширину объекта.

Свойство Height – определяет высоту объекта.

Свойство Left – указывает расстояние от объекта до левой границы формы.

Свойство Top – указывает расстояние от объекта до верхней границы формы.

*5.3. **События. Реакция на события. Окно кода программы.***

События устанавливаются в окне Инспектора Объектов на странице Events (События). Рассмотрим первые четыре события:

OnClick - событие щелчок мыши.

OnDbClick - событие двойной щелчок мыши.

OnClose - закрытие окна формы.

OnCreat – возникает после создания объекта, но до его появления на экране.

Для задания события нужно выделить данный компонент, перейти в Инспекторе Объектов на страницу Events и дважды щелкнуть мышью на нужном событии. После выбора события автоматически открывается окно кода программы.

Окно кода программы предназначено для создания и редактирования текста программы. Этот текст составляется по определенным правилам. Совокупность правил записи текста называется языком программирования. В системе программирования Delphi используется язык программирования Object Pascal.

После задания события в коде программы автоматически создается процедура обработки события, внутри которой описывается реакция на событие. Заголовок процедуры формируется следующим образом:

Procedure *TF1. Button1Click (Sender:TObject);*

{событие нажатие мыши на компоненте Button1, расположенной на форме F1}

Begin

End;

Между Begin End указывается **реакция на события - перечень совершаемых действий**.

5.4. ***Сохранение, компилирование, запуск программы.***

File / Save as …– сохраняет программу с расширением .pas. Имя файла желательно задавать английскими буквами. Например, sdr.pas.

File / Save project as … – сохраняет проект, работающий с программой, с расширением .dpr. Имя проекта желательно формировать следующим образом: римя файла.dpr. Напрмер, psdr.dpr.

Project/Compile – для компиляции программы и обнаружения ошибок. В случае если отладчик обнаружит ошибку, он выделит эту строку красным цветом. В таком случае исправьте ошибку и запустите команду заново.

Project/Information – показывает итоги компиляции, надпись Successfully свидетельствует о том, что ошибки не обнаружены.

Run/Run. – запуск программы на исполнение.

Run/ Reset – прекращение работы программы.

Тема 6. **Обзор компонент**

6.1. ***Компоненты страницы Standard.***

На первой странице Палитры Компонент размещены 14 объектов определенно важных для использования. Мало кто обойдется длительное время без кнопок, списков, окон ввода и т.д.

Курсор – не компонент, просто пиктограмма для быстрой отмены выбора какого-либо объекта.

TMainMenu – горизонтальной меню (строка меню).

TPopupMenu – всплывающее меню, появляется по щелчку правой кнопки мыши на объекте.

TLabel – метка**,** служит для отображения текста на экране.

TEdit – однострочное окно ввода, вывода.

TMemo – многострочное окно ввода и вывода, имеет основные функции редактора. TMemo имеет ограничения на объем текста в 32Кб, это составляет 10-20 страниц.

TButton – командная кнопка.

TCheckBox – флажок. Если команда отмечена флажком, это означает, что она выбрана.

TRadioButton – радиокнопка, позволяет выбрать только одну опцию из нескольких.

TListBox – прокручиваемый список.

TCobmoBox напоминает TListBox с полем ввода сверху.

TScrollBar – полоса прокрутки.

TGroupBox – группа элементов, используется для группировки нескольких связанных по смыслу компонентов.

TRadioGroup – группа радиокнопок.

TPanel – панель, используется в декоративных целях для объединения нескольких компонент. Обладает эффектом «вдавленности» и «выпуклости».

*6.2. **Компоненты страницы Additional.***

На странице Additional размещены объекты, позволяющие создать более красивый пользовательский интерфейс программы.

TBitBtn – кнопка вроде TButton, однако на ней можно разместить картинку (glyph).

TSpeedButton – кнопка для панели инстументов – панели быстрого доступа к командам (SpeedBar).

TTabSet – горизонтальные закладки. Обычно используется вместе с

TNoteBook – используется для создания многостраничного диалога, на каждой странице располагается свой набор объектов.

TTabbedNotebook – многостраничный диалог со встроенными закладками.

TMaskEdit - аналог TEdit, но с возможностью форматированного ввода.

TOutline – используется для представления иерархических отношений связанных данных. Например - дерево директорий.

TStringGrid – служит для представления текстовых данных в виде таблицы.

TDrawGrid – служит для представления данных любого типа в виде таблицы.

TImage – отображает графическое изображение на форме.

TShape – служит для отображения простейших графических объектов на форме: окружность, квадрат и т.п.

TBevel – элемент для рельефного оформления интерфейса.

THeader – элемент оформления для создания заголовков с изменяемыми размерами для таблиц.

TScrollBox – позволяет создать на форме прокручиваемую область с размерами большими, нежели экран. На этой области можно разместить свои объекты.

*6.3. **Некоторые компоненты дополнительных страниц.***

Страница Dialog. На странице Dialogs представлены компоненты для вызова стандартных диалогов Windows: открытие файла, сохранение файла, настройка шрифта, выбор цвета, печать, поиск строки, поиск с заменой.

Страница System представляет набор компонент для доступа к некоторым системных сервисам типа таймер, DDE, OLE и т.п.

Страницы Data Access, Data Controls содержат компоненты, которые облегчают доступ к базам данных и содержат стандартные интерфейсные элементы при работе с базами данных.

Страницы Internet обеспечивают средства связи программы с глобальной компьютерной сети Internet.

ЧАСТЬ3. ЯЗЫК ПРОГРАММИРОВАНИЯ OBJECT PASCAL

Тема 7. Структура программ

Любая программа в Delphi состоит из файла проекта (файл с расширением dpr) и одного или нескольких модулей (файлы с расширением pas). Каждый из таких файлов описывает программную единицу Object Pascal.

Структура любой программной единицы (проект, модуль, подпрограмма) состоит из заголовка, раздела описания данных, выполняемой части (раздел операторов).

7.1. ***Структура проекта***

Файл проекта представляет собой программу, написанную на языке Object Pascal, и к которой подключаются созданные модули. Проект автоматически создается Delphi и содержит лишь несколько строк. Чтобы увидеть их, выберите команду *Project/View source*.

```
program Project1;
uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1};
{$R *.RES}
begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

В окне жирным шрифтом выделяются зарезервированные слова, а курсивом – комментарии.

Программа начинается с заголовка: **program** имя программы. За заголовком следует раздел описаний данных. С помощью слова **uses** (использовать) сообщается о модулях, которые подключаются к проекту. Строки

```
Uses Forms,
  Unit1 in 'Unit1.pas' {Form1};
```

указывают, что, помимо файла проекта, в программе должны использоваться модули Forms и Unit1. Модуль Forms является стандартным, а модуль Unit1 – новый, его следует подключить к программе с именем ‘Unit1.pas’ и именем связанного с модулем файла формы {Form1}.

Выполняемая часть – тело программы – начинается со слова **begin** и заканчивается словом **end**. Тело программы состоит из операторов языка Object Pascal. В приведенном примере три оператора:

```
Application.Initialize;
 Application.CreateForm(TForm1, Form1);
 Application.Run;
```

Каждый из них реализует обращение к одному из методов объекта *Application*. В объекте Application собраны данные и подпрограммы, необходимые для нормального функционирования Windows-программы в целом. Метод *Initialize* заставит процессор перейти к выполнению большой подпрограммы, написанной разработчиками Delphi. После выполнения этой подпрограммы процессор перейдет к следующей строке программы, методу *CreateForm* – создание и показ на экране окна главной формы. Метод Run реализует бесконечный цикл получения и обработки событий, в том числе описанных в созданном подключаемом модуле.

7.2. ***Структура модуля.***

Модуль − это группа связанных друг с другом функций и процедур. Любой модуль имеет следующую структуру:

заголовок (**unit)**,

секция интерфейсных объявлений -раздел описания (**interfase)**,

секция реализаций - выполняемый раздел (**implementation)**.

В секции интерфейсных объявлений описываются программные элементы, которые будут «видны» другими программными модулями, а в секции реализаций раскрывается механизм работы этих элементов.

unit имя модуля;

заголовок имя модуля должно совпадать с именем файла

interfase

uses ... список модулей, с которыми устанавливается связь

type ... объявления используемых типов;

private … закрытый − объявления переменных, доступных только внутри модуля

protected … открытый − объявления переменных, доступных за пределами модуля

function ...

procedure ...

объявления функций и процедур;

implementation

реализация процедур и функций, описанных в первой части модуля

```
procedure …
begin ….
end;
function …
begin …
end;
……
end.
```

7.3. ***Структура подпрограмм.***

Подпрограммами в Object Pascal являются процедуры и функции.

При описании подпрограммы используются формальные параметры – переменные, непосредственно участвующие в вычислении результата подпрограммы. При обращении к подпрограмме применяются фактические параметры, значения которых присваиваются формальным параметрам.

Функцию используют, если результатом подпрограммы является одно значение. Тип значения фиксируется в описании функции.

Структура заголовка функции следующая:

function имя (список формальных параметров: тип параметров):тип результата функции;

Имя функции является результатом (выходным данным), описание входных данных – список параметров.

Обращение к функции осуществляется:

Идентификатор:= имя функции (список фактических параметров);

Процедуру используют, если результатом подпрограммы является любое количество значений.

Структура заголовка процедуры следующая:

procedure имя(список входных формальных параметров: тип параметров,
var список выходных формальных параметров: тип параметров);

Обращение к процедуре осуществляется:

Имя процедуры (список входных фактических параметров,
список выходных фактических параметров);

Структура любой подпрограммы (процедуры или функции), как и все программные единицы, состоит из трех разделов: заголовок подпрограммы, раздел описания данных, раздел операторов.

заголовок подпрограммы

раздел описания данных:

Раздел константа.

const

идентификатор 1 = const1;

идентификатор 2 = const2;

Раздел типов.

type

идентификатор типа 1 = описание типа 1;

идентификатор типа 2 = описание типа 2;

Раздел переменных.

var

список 1 переменных: тип 1 переменных;

список 2 переменных: тип 2 переменных;

Раздел функций и процедур.

Раздел операторов

begin

оператор 1;

оператор 2;

...

end.

Тема 8. **Элементы языка**

8.1. ***Основные понятия языка.***

Алфавит языка Object Pascal включает буквы, цифры, специальные символы, пробелы, зарезервированные (ключевые слова).

Цифры – арабские цифры от 0 до 9.

Ключевые (зарезервированные) слова – это последовательность символов, имеющая в данном языке особый установленный смысл. Ключевые слова нельзя использовать в качестве идентификаторов. Ключевые слова выделяются жирным шрифтом.

Например, в Object Pascal имеются следующие зарезервированные слова:

And	**Else**	**Nil**	**String**
Array	**End**	**Not**	**Then**
Begin	**for**	**Object**	**To**
Class	**File**	**Of**	**Type**
Const	**Function**	**Or**	**Until**
Construction	**If**	**Out**	**Unit**
Destruction	**Implementation**	**Procedure**	**Uses**
Div	**In**	**Program**	**Var**
Do	**Initialization**	**Record**	**While**
Downto	**Interface**	**Repeat**	**With**
	Mod	**Set**	**xor**

Идентификаторы - это имена констант, переменных, типов, модулей, объектов и т.п. Идентификаторы могут иметь различную длину, но значащими являются только первые 63 символа. Идентификаторы всегда начинаются буквой, за которой могут следовать буквы и цифры, не содержат пробелов и специальных символов.

Оператор – минимальная структура в программе, производящая законченное действие. Оператор содержит ключевое слово, которое определяет его смысл.

8.2. ***Данные. Структура типов данных. Простые типы данных.***

Данные в Object Pascal бывают:

константы (сохраняют свое значение в течение работы всей программы);

переменные (могут изменять значение в результате работы программы);

значение функций или выражений.

Любые данные характеризуются своими типами. Тип определяет:

множество допустимых значений, которые может иметь то или иное данное;

множество допустимых операций, которые применимы к данному;

объем памяти, выделяемый для хранения данного.

Object Pascal характеризуется разветвленной структурой типов данных (рис. 8.1).

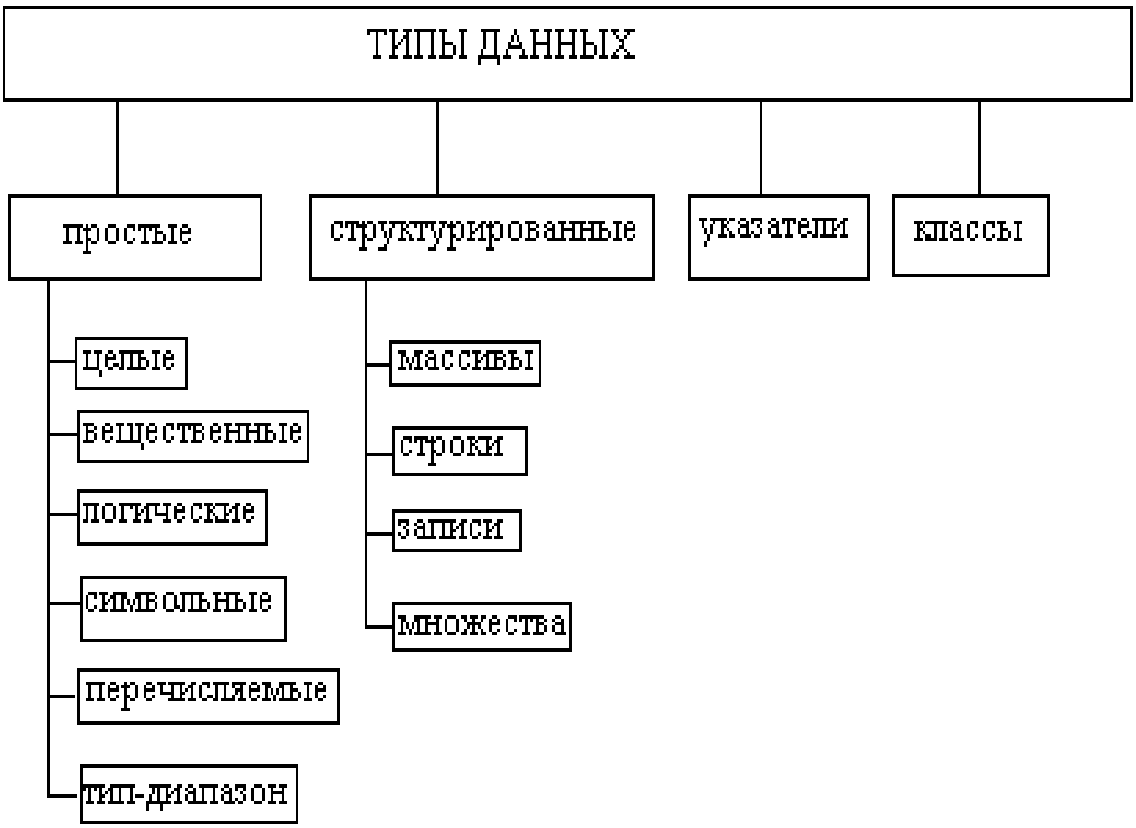


Рис.8.1. Типы данных Object Pascal.

Данные простого типа хранят в себе только одно значение. К простым типам относятся целые, вещественные, логические, символьные, перечисляемые, тип-диапазон.

Целые типы данных используются для представления целых чисел. Есть несколько различных целых типов, которые могут хранить целые значения (табл. 8.1). Различные целые типы имеют существенно различные диапазоны хранимых значений, затраты памяти растут с ростом допустимого диапазона значений.

Таблица 8.1. Целые типы данных.

Тип	**Диапазон значений**	**Требования к памяти**
Byte	[0; 255]	1 байт
Word	[0; 65535]	2 байт
Shorting	[-128; 127]	1 байт
Smallint	[-32768; 32767]	2 байт
Integer	[-2147483648; 2147483647]	4 байт
Cardinal	[0; 2147483647]	4 байт
Longint	[-2147483648; 2147483647]	4байт

Вещественные типы данных предназначены для хранения чисел, имеющих дробную часть. В табл. 8.2 представлены несколько различных вещественных типов данных.

Таблица 8.2. Вещественные типы данных.

Тип	**Диапазон значений**	**Требования к памяти**
Real	$[2.9*10^{-39}; 1.7*10^{38}]$	6 байт
Single	$[1.5*10^{-45}; 3.4*10^{38}]$	4 байт
Extended	$[3.4*10^{-4932}; 1.1*10^{4392}]$	10 байт

Логический тип данных – один из простейших и часто используемых. Примером логического типа является ***Boolean*.** Переменные типа Boolean представляют собой логические значения True или False.

Символьный тип данных. Значениями символьного типа являются множество всех символов компьютера (буквы, цифры, арифметические знаки, орфографические знаки, специальные символы). Каждому символу приписывается код – целое число в диапазоне [0;255]. Код необходим для внутреннего представления символа.

Для кодировки в Windows используется код ANSI (назван по имени American National Standard Institute – Американского института стандартизации).

Примером символьного типа данных является тип ***Char***, использующийся для хранения одного символа и в памяти отводится 1 байт.

К типу Char применимы функции:

Ord(символ) – возвращает код указанного символа.

Chr(код) – возвращает символ, соответствующий указанному коду.

Перечисляемый тип задается перечислением тех значений, которые он может получать. Каждое значение именуется некоторым идентификатором и располагается в списке, обрамленном круглыми скобками. Применение перечисляемых типов делает программы нагляднее. Например, если в программе идут данные по месяцам, то можно использовать перечисляемый тип:

```
Type  TypeMec = (jan, feb,mart, apr, may,jun,jul,augyct,sep,oct,nov,dec);
Var  Месяц : Тип месяца;
Begin
…….
If месяц = augyct then label1.caption:=’Хорошо бы поехать к морю!’;
……..
end;
```

Тип-диапазон есть подмножество любого простого типа. Он задается границами своих внутренних значений:

<минимальное значение> . . <максимальное значение>

Например:

```
Var
  Date : 1..31;
  Month : 1..12;
  Bykv : ‘A’.. ‘Z’;
```

*8.3. **Математические выражения, функции и операции.***

Математические выражения – это операции с числами, переменными, значениями функций.

Порядок выполнения операций – слева направо, при этом сначала выполняются действия в скобках, после умножение, деление, арифметические функции, а дальше сложение, вычитание. Числа записываются при помощи цифр, причем целая часть от дробной отделяется точкой.

Например: 4.5*(b-2.2)/(c+5.1).

Числа могут быть представлены в показательной форме в виде:

1.76E-3 – запись соответствует $1.76*10^{-3}=0.00176$

3.765E4 – запись соответствует $3.765*10^{4}= 37650$

Основные арифметические операции и функции, используемые при записи выражений, представлены в табл. 8.3.

Таблица 8.3.**Основные арифметические операции и функции.**

Операция	**Действие**	**Тип операндов**	**Тип результата**	**Пример**
+	Сложение	Любой	Соответствует типу операндов	4+5 = 9
-	Вычитание			9 - 4 = 5
*	Умножение			2 * 2 = 4
/	Деление	Любой	Вещественный	10/4 = 2.5
Div	Целочисленное деление	Целый	Целый	9 div 4 = 2
Mod	Остаток от деления	Целый	Целый	9 mod 4 = 1
Abs	Модуль числа	Любой	Соответствует типу операнда	Abs(-5.3)=5.3
SQR	Возведение в квадрат	Любой	Соответствует типу операнда	SQR(4)=16
SQRT	Квадратный корень	Любой	Вещественный	SQRT(78)=2.97
Cos	Косинус	Любой	Вещественный	Cos(4)=0.997
Sin	Синус	Любой	Вещественный	Sin(4)=0.069
Exp	Экспонента	Любой	Вещественный	Exp(4)
Ln	Натуральный логарифм	Любой	Вещественный	Ln(4)=1.38
Round	Округление до ближайшего целого	Вещественный	Целый	Round(4.5)=5
Trunc	Возвращает целое число путем отбрасывания дробной части	Вещественный	Целый	Trunc(4.5)=4

8.4. ***Операции отношения. Логические операции.***

Выражения можно сравнивать при помощи операции отношения:

= *равно;*

<> *неравно;*

< *меньше;*

> *больше;*

<= *меньше или равно;*

>= *больше или равно.*

Результатом применения операций отношения является логическое данное True или False. Например, результатом выражения 3>1 является True, а результатом отношения 3<>3 – False.

Сложные логические выражения составляются с использованием логических операций:

Not *– логическое НЕ;*

And *– логическое И;*

Or *– логическое ИЛИ;*

Xor *– исключительное ИЛИ.*

Логические операции применимы к логическим данным и дают результат логического типа. Можно привести следующий пример:

```
X > 5 and X <= 10,
```

где *X > 5* – операнд1 логического типа, *X* <= 10 – операнд2 логического типа, *and* – логическая операция. Результатом всего выражения будет логический тип.

Ниже в таблицах приведены правила формирования логического результата для выражения вида *операнд1* ***логическая операция*** *операнд2.*

Таблица 8.4. Логическая операция **And** над данными логического типа.

Операнд1	**Операнд2**	**Результат выражения о*перанд1 and операнд2***
True	True	True
False	True	False
True	False	False
False	False	False

Таблица 8.5. Логическая операция **Or** над данными логического типа.

Операнд1	**Операнд2**	**Результат выражения** *операнд1* ***or*** *операнд2*
True	True	True
False	True	True
True	False	True
False	False	False

Таблица 8.6. Логическая операция **Xor** над данными логического типа.

Операнд1	**Операнд2**	**Результат выражения о*перанд1 xor операнд2***
True	True	False
False	True	True
True	False	True
False	False	False

Таблица 8.7. Логическая операция **Not** над данным логического типа.

Операнд1	**Результат выражения not (о*перанд)***
False	True
True	False

Тема 9. **Операторы языка**

Оператор – минимальная структура в программе, производящая законченное действие. Оператор содержит ключевое слово, которое определяет его смысл.

9.1. ***Оператор присваивания.***

Это наиболее часто используемый оператор языка. Синтаксис оператора присваивания:

Идентификатор:= выражение;

Символы «:=» всегда пишутся слитно, без разделяющих пробелов, хотя перед двоеточием и после знака равенства можно для лучшей читаемости программы вставлять пробелы. Как и любой другой оператор языка, оператор присваивания завершается точкой с запятой. Переменные и результат выражения должны быть одного типа.

9.2. ***Составной оператор.***

Это последовательность произвольных операторов программы, заключенная в операторные скобки – **begin … end**.

Составной оператор может состоять из любого количества операторов:

```
begin
   оператор1;
   оператор2;
   ...
   операторN;
end;
```

Среди вложенных операторов могут быть и другие составляющие операторы:

```
Begin
    ……..
    Begin
        ……..
        begin
            ……..
            ……..
        end;
        ……..
    End;
    ……..
end;
```

9.3. ***Условный оператор.***

Условный оператор (ветвление) позволяет проверить некоторое условие и в зависимости от результатов проверки выполнять то или иное действие.

Условие в языке Object Pascal – это выражение логического типа (*BOOLEAN*), которое может принимать одно из двух значений: "истина" (*TRUE*) или "ложь" (*FALSE*). Условие записывается с помощью операций отношения и логических операций.

Существует два типа ветвления: *полное* и *неполное*.

Структура полного ветвления имеет следующий вид:

If <условие> **then** <оператор1> **else** <оператор 2>;

где **if, then, else** – ключевые слова (если, то, иначе).

Условный оператор работает по следующему алгоритму. Сначала вычисляется условие. Если результат условия *True* (истина), то выполняется оператор1, а оператор2 пропускается. Если результат условия False (ложь), то выполняется оператор2, а оператор1 пропускается. Например:

```
Var x, y, max: integer;
Begin
  ……..
  If  x > max  then y:= max  else y:=x;
…….
End;
```

При выполнении этого фрагмента переменная *y* получит значение переменной *max*, если значение переменной *x* больше *max;* в противном случае *y* станет равно *max*.

Сложные логические выражения составляются с использованием логических операций **and, or, not, xor**. Например:

```
If  (x > max) and ( y>max)  then y:= max  else y:=x;
```

Часто в случае истинности или ложности условия выполняется более одного оператора, в таком случае используется составной оператор. Например, нахождение максимального из двух чисел a и b и вычисления их разности:

```
Var a,b,c: real; str: string;
Begin
……..
If a>b
     Then begin
          Str :='a больше b';
          C := a-b;
          End
     Else  begin
           Str :='b больше a';
          C := b-a;
End;
```

Структура неполного ветвления имеет следующий вид:

if условие **then** оператор;

Неполный условный оператор работает по следующему алгоритму. Сначала вычисляется условие. Если условие принимает значение TRUE, то выполняется оператор, иначе никаких действий не происходит.

9.4. ***Оператор выбора.***

Оператор выбора позволяет выбрать одно из нескольких возможных продолжений программы. Параметром, по которому осуществляется выбор, служит **ключ выбора** – выражение любого целого, символьного и логического типа.

Структура оператора выбора такова:

```
case  <ключ выбора> of
    <константа выбора>: оператор 1;
    <константа выбора>: оператор 2;
    .....
    <константа выбора>: оператор N;
    else оператор
end;
```

Оператор выбора работает следующим образом. Вычисляется значение выражения <ключ выбора>, затем полученное значение последовательно сравнивается с константами выбора из списка констант. Если значение выражения совпадает с константой из списка, то выполняется соответствующая этому списку последовательность операторов, и на этом выполнение инструкции **case** завершается. Если значение выражения не совпадает ни с одной константой из всех списков, то выполняется последовательность операторов, идущих после **else.**

Константа выбора может содержать не одно, а несколько значений, разделенных запятой, или представлять диапазон значений, в котором начальное и конечное значения разделены двумя точками.

Например, школьник учится в N классе. В зависимости от класса необходимо определить школьную ступень образования (начальное, среднее, старшие классы).

```
Var N: BYTE; Str:string;
begin
……..
case N of
  1..4: str := ’начальная школа’;
  5..9 : str := ‘среднее звено’;
  10,11: str := ‘старшие классы’;
   else str := ‘неверно указан номер класса’;
end;
```

Иногда веточка **else** в операторе выбора может отсутствовать.

9.5. ***Циклические операторы.***

Циклические операторы содержат повторяющиеся команды.

Циклические операторы бывают трех видов: счетный цикл (цикл с параметром), цикл с предусловием, цикл с постусловием.

Структура оператора цикла с предусловием следующая:

while условие **do** оператор;

Здесь **while**, **do** – зарезервированные слова (*пока [выполняется условие], делать*).

Циклический оператор с предусловием работает по следующему алгоритму. Первоначально проверяется условие. Если оно истинно, то выполняется тело цикла (оператор). Затем снова проверяется условие и т.д. Если ус-

ловие ложно, то цикл завершается и выполняется оператор, стоящий непосредственно после цикла.

Если тело цикла состоит из нескольких операторов, то используется составной оператор.

Структура оператора цикла с постусловием следующая:

repeat тело цикла **until**;

Циклическая структура цикла с постусловием работает по следующему алгоритму: сначала выполняется тело цикла (операторы), потом проверяется условие. Если условие ложно, то снова выполняются операторы. И так до тех пор, пока условие не примет значение *TRUE*.

Блок-схема счетного цикла (с параметром) следующая:

for <параметр цикла**>:=<**нач.знач.**> to <**кон**.** знач**>.do <**оператор**>**;

Циклическая структура счетного цикла работает по следующему алгоритму: параметру цикла присваивается начальное значение и выполняется тело цикла. Затем параметру цикла присваивается следующее значение, и снова выполняется тело цикла. И так до тех пор, пока не будут перебраны все значения параметра цикла. Каждый раз параметр цикла увеличивается на единицу.

Другая структура счетного оператора:

for <параметр цикла**>:=<**старш. знач**>.downto <**млад. знач.**> do<** оператор**>**;

Циклическая структура данного счетного цикла работает по следующему алгоритму: параметру цикла присваивается начальное (старшее) значение и выполняется тело цикла. Затем значение параметра уменьшается на единицу, и снова выполняется тело цикла. И так до тех пор, пока не будут перебраны все значения параметра цикла до конечного (младшего) значения.

Тема 10. **Структурированные данные**

Данные структурированного типа хранят в себе множество значений. Структурированные типы бывают четырех видов: массивы, строки, записи, множества. В последующих темах будет рассмотрена подробная работа с каждым типом, сейчас остановимся только на вопросе описания структурированных типов данных.

10.1. ***Массивы.***

Массивы – упорядоченный набор однотипных элементов. Все элементы должны быть одного и того же типа.

В Object Pascal возможна работа с несколькими видами массивов – одномерными, двумерными, динамическими.

В одномерном массиве компоненты представлены в виде одной строки (одного столбца). Все компоненты одного типа. Каждый компонент имеет свой индекс.

Описание одномерного массива возможно двумя способами:

1. В разделе описания типов:

```
type
  <имя типа массива> = array [<тип индекса>] of <тип компонент>;
var
  <имя массива>: <имя типа массива>;
```

2. В разделе описания переменных:

```
var
  <имя массива>: array [<тип индекса>] of <тип компонент>;
```

Обычно в качестве индексного типа используется тип-диапазон, в котором задаются границы изменения индексов. Например:

```
Type
  Tnumber = array [0..19] of  integer;
Var number: Tnumber;
```

или

var *number:* ***array*** *[0..20]* ***of*** *integer;*

К конкретному элементу массива можно обратиться по имени и индексу. При этом индекс записывается в квадратные скобки, например: number**[0]**, number**[2]**, ..., number**[19]**.

Двумерные массивы представляют собой **матрицу**. Элементы в матрице располагаются по строкам и столбцам.

Описание двумерного массива:

type

<имя типа>= **array** [<тип инд. строк, тип инд. столбцов>] **of** <тип компонент>;

var

<имя массива>: <имя типа массива>;

или

var

<имя массива>:**array** [<тип инд. строк, тип инд. столбцов>] **of** <тип компонент>;

Чтобы обратиться к конкретному элементу матрицы, надо указать в квадратных скобках, через запятую индекс строки, индекс столбца. В Object Pascal можно одним оператором присваивания передать все элементы одного массива другому. Например:

```
Var
A,B: array [1 .. 5, 1..4] of  real;
Begin
   A[1,1]:=1;
   A[1,2]:=2;
   A[1,3]:=4;
   B:=A;
End;
```

В версии Delphi 4 впервые введены динамические массивы. При объявлении таких массивов в программе не следует указывать границы индексов:

```
Var A: array of integer;
    B: array of array of Char;
    C: array of array of array of real;
```

В этом примере динамический массив A имеет одно измерение, массив B – два, массив C – три. Распределение памяти и указание границ индексов по каждому измерению динамических массивов осуществляется в ходе выполнения программы путем инициализации массива с помощью функции *SetLength*. Например, при инициализации одномерного массива A:

```
SetLength (A,3);
```

одномерный динамический массив получит память для размещения трех значений. Нижняя граница индексов всегда равна 0, поэтому верхней границей индексов будет 2.

Для освобождения памяти, занимаемой массивом, достаточно присвоить идентификатору значение NIL (или использование процедуры Finalize):

```
A:=nil;
Finalize(B);
```

При изменении длины уже инициированного массива сначала инициируется новый массив большей памяти. Затем элементы старого массива переносятся в новый, после чего высвобождается память, выделенная прежним массивом.

В многомерных массивах сначала устанавливается длина его первого измерения, затем второго и т.д. Например:

```
Var B: array of array of Char;
Begin
  SetLength (B,3);    SetLength (B[0],3);
  SetLength (B[1],3);  SetLength (B[2],3);
  ……
end;
```

10.2. ***Строки.***

Строки – последовательность символов, заключенная в апострофы. В Object Pascal имеются строковые типы:

короткая строка *ShortString*,

длинная строка *String,*

широкая строка *WideString*,

нуль-терминальная строка *Pchar.*

Наиболее часто используемым строковым типом является длинная строка, которая задается с помощью зарезервированного слова

String[n], где n <=255.

Символу в строке индексированы, начиная с индекса 1. К отдельному символу в строке можно обратиться при помощи индексации. Например: *S:= ‘дисциплина’,* тогда S[1] равен ‘д’, S[5] равен ‘и’.

Две строки можно сравнивать друг с другом с помощью операций отношения. Сравнение идет слева направо по таблице кодов: сначала первые символы, если он совпадают – сравниваются следующие и т.д.

Процедуры и функции для работы со строками.

Преобразования числа в строку

STR(x, st: string) – преобразует числовое значение x в строку символов st.

IntToStr (x: integer) – преобразует целое число x в строку.

FloatToStr (y: real) – преобразует вещественное число в строку.

Преобразования строки в число

StrToInt (st:string) – преобразует строку в целое число.

StrToFloat (st:string) – преобразует строку в вещественное число.

Val(st:string, x, code:integer) – преобразует величину st в числовую величину вещественного или целочисленного типа x. Если преобразование успешно, то переменной cod присваивается значение 0; если обнаруживается ошибочный символ, то значение x не определяется, а переменной cod присваивается значение номера ошибочной позиции.

Обработка строковых величин

LENGTH(st: string) - выдает длину строки st. Например:

length('delphi')→6;

length('object pascal')→13;

CONCAT(st1, st2, ..., stn) - склеивание строк st1, st2, ..., stn, аналогично операции '+'. Например:

S:=’информационные’;

T:=’технологии’;

concat(s, ' ' ,t) →’информационные технологии’.

POS(st1: string, st: string) – отыскивается вхождение подстроки st1 в строку st. Результат – целое число, равное номеру позиции, с которой в строке st начинается строка st1. Если подстрока st1 не найдена, то значение функции равно 0. Например:

pos('rb',s)→3; pos('tar',s)→0.

COPY(st:string, m:integer, n:integer) – функция копирует из строки st подстроку, длиной в n символов, начиная с позиции m. Например:

L:=’паровоз’;

copy(L,1,3) → ’пар’; copy(L,4,3)→’воз’.

DELETE(st: string, m: integer, n:integer) – удаляет из строки st подстроку длиной в n, начиная с m символа. Например:

st:='Object pascal'; delete(st,7,7)→’Object’.

INSERT(st1: string, st:string, m: integer) – вставляет строку st1 в строку st, начиная с m позиции. Если m больше длины строки st, то строка st1 просто присоединяется к st1. Например:

st:='abcd'; insert('xx',st,3) → 'abxxcd';

st1:='ef'; insert(st1,st,4) → 'abcdef'.

10.3. ***Записи.***

Записи – это структура данных, состоящая из фиксированного количества компонентов, называемых полями записи. Поля записи могут быть различного типа.

Структура объявления типа записи следующая:

```
Type
Имя типа = record
   Поле1: тип;
   Поле2: тип;
   …..
end;
```

Например, если программа работает со списком студентов, то мохно использовать следующий тип записи:

```
Type
TStudent = record
             FIO:string;
           Gryppa: string[5];
          Let:byte;
           end;
var  student : Tstudent;
```

К каждому полю записи можно обратиться, используя составное имя:

Имя переменной. имя поля

Например:

```
Student.fio:='Иванов И.И.';
Student.gryppa:= 231;
Student.let:=20;
```

10.4. ***Множества.***

Множества – это набор однотипных однотипных, логически связанных друг с другом объектов.

Над множествами можно проделывать операции: пересечение множеств, объединение множеств, разность множеств, определение эквивалентности.

Описание типа множества имеет вид:

<Имя типа> = **set of** <базовый тип>;

где базовый тип – тип-диапазон возможных значений.

Пример описания переменных типа множества:

```
Var
  S1,S2: set of 'A'..'Z';
  B1,B2: set of  0..9;
 Begin
S1:=['B','D','R','T'];
S2:=['A','B','C','R','K']
B1:=[3,6,2,1,8,4];
B2:=[1,3,5,7];
…….
End;
```

Над множеством определены следующие операции:

* пересечение множеств – результат содержит элементы, общие для обоих множеств. Например:

```
S1*S2  ->  ['B','R'];
B1*B2 -> [1,3];
```

+ объединение множеств – результат содержит элементы первого множества, дополненные недостающими элементами из второго. Например:

```
S1+S2  -> ['B','D','R','T','A','C','K'];
```

- разность множеств – результат содержит элементы из первого множества, которые не принадлежат второму. Например:

B1-B2 -> [6,2,8,4];

S2-S1 -> [‘A’,’C’,’K’];

= проверка эквивалентности, возвращает True, если оба множества эквивалентны.

<= проверка вхождения, возвращает True, если первое множество включено во второе.

<элемент> IN <множество> – проверка принадлежности элемента множеству. Например:

3 **in** B1 -> True;

5 **in** b1 -> false;

Include (<множество>,<элемент>) – включает новый элемент в множество.

Тема 11. **Файлы**

Файлы в Object Pascal делятся на три категории: типизированные, нетипизированные, текстовые.

Каждый из типов файлов обладает специфическими особенностями, однако порядок работы с ними фактически один и тот же.

11.1. ***Общие принципы работы с файлами.***

Последовательность работы со всеми типами файлов на Pascal следующая:

1) объявление *файловой переменной* соответствующего типа;
2) связывание такой переменной с *именем* дискового файла (как правило, указывается полное имя – путь и имя);
3) открытие/создание файла – файл может быть открыт как для чтения, так и для записи; для некоторых типов файлов возможно открытие одновременно для чтения и записи;
4) непосредственно обмен информацией – чтение/запись данных из/в файл. При этом возможны *перемещения* по файлу – изменение позиции чтения/записи;
5) закрытие файла – освобождение ресурса ОС.

11.2. ***Текстовые файлы.***

Текстовые файлы состоят из строк (разной длины), разделенных парами символов возврата каретки – Enter и пробел (#13#10).

Описание файловой переменной текстового файла осуществляется следующим образом:

<имя переменной>: **text**.

Для всех типов файлов операция сопоставления файловой переменной с именем файла на диске однотипна:

Assign(F:text; S: String); – связывает файловую переменную F с именем внешнего файла S.

Function IOResult: Integer; – возвращает код ошибки последней операции ввода-вывода. Успешное завершение операции сигнализируется кодом ошибки 0. Функция IOResult считается операцией ввода-вывода и всегда завершается успешно.

ReWrite(F: text**); –** создает и открывает для чтения/записи *новый* (пустой) внешний файл, с именем, которое указывалось при связывании файловой переменной.

Reset(**Var F** :**text); –** открывает *существующий* внешний файл для чтения/записи. Указатель чтения/записи устанавливается на начало файла.

Append(Var F: Text); – открывает для дополнения текстовый файл. Указатель записи устанавливается на конец файла.

EoF(File):Boolean; – возвращает True, если текущий указатель стоит за концом файла (т. е. записей больше прочитать нельзя).

Read(F:text,V1,V2,V3, …:string); – читает одну или несколько записей, начиная с текущей позиции чтения.

ReadLn(F:text,V1,V2,V3, …:string); – читает одну или несколько записей, начиная с текущей позиции чтения, и переводит указатель на следующую строку.

Write(F, V1,V2,V3, …: string); - записывает в файл одну или несколько записей, начиная с текущей позиции указателя записи.

WriteLn(F, V1,V2,V3, …: string); - записывает в файл одну или несколько записей, начиная с текущей позиции указателя записи, и переводит указатель на следующую строку.

Close(**Var F: text)**;- закрывает открытый файл.

Например, подпрограмма для записи информации в файл может быть следующая:

```
var f: text; s: string;
begin
s:='C:\Мои документы\t1.txt';
Assign(F,s);
if IOResult<>0
  then s:='ошибка ввода-вывода'
  else begin
        ReWrite(F);
        Write(F,'строка1');
        Write(F,'строка2');
        Close(F);
       end;
end;
```

Подпрограмма для чтения информации из файла может быть следующая:

```
var f: text; s: string;
begin
s:='C:\Мои документы\t1.txt';
Assign(F,s);
if IOResult<>0
  then s:='ошибка ввода-вывода'
  else begin
         ReWrite(F);
           while EOF<>true do begin
                                Write(F,s);
                                ……  end;
        Close(F);
          end;
 end;
```

*11.3.**Типизированные файлы.***

Типизированные файлы состоят из записей – участков одинакового размера и типа, расположенных последовательно.

Описание файловой переменной типизированного файла осуществляется следующим образом:

<имя переменной>: **file of** <тип_элементов>;

Открытие, запись/чтение информации, закрытие типизированного файла осуществляются аналогично текстовому файлу.

Записи типизированного файла нумеруются с нуля. Постоянство размера каждой записи позволяет производить *позиционирование* – установку позиции чтения/записи – на любую из них.

Seek(F:text;N:Longint); – помещает текущий указатель чтения/записи файловой переменной F на запись номер N.

Следует следить, чтобы позиция, на которую устанавливался указатель чтения/записи, была правильной, т. е. не выходила за пределы файла.

Определить текущее значение файлового указателя можно с помощью функции **FilePos(F),** а общее количество записей – **FileSize(F).**

Таким образом, существует возможность *произвольного* доступа к любой из существующих записей типизированного файла. Естественное назначение типизированных файлов – сохранение/восстановление множества однотипных данных.

*11.4.**Нетипизированные файлы.***

Нетипизированные файлы отличаются тем, что для них не указан тип компонентов, это позволяет работать с различными компонентами.

Описание файловой переменной типизированного файла осуществляется следующим образом:

<имя переменной>: **file**;

Открытие нетипизированных файлов осуществляется аналогично текстовым файлам, но читаются и пишутся они не записями, а блоками. Принципиальным различием между записью и блоком является то, что размер блока указывается *при открытии* нетипизированного файла.

ReWrite(F: text; RecSize:Word**);** – создает и открывает для чтения/записи *новый* (пустой) внешний файл, с именем, которое указывалось при связывании файловой переменной. Дополнительный параметр указывает *размер записи*; по умолчанию (при отсутствии его для нетипизированных файлов) размер записи принимается равным `128` байтам.

ReSet(Var F:**File**; RecSize:Word); – открывает *существующий* внешний файл для чтения/записи. Указатель чтения/записи устанавливается на начало файла. Дополнительный параметр RecSize указывает размер блока данных.

Чтение и запись в нетипизированные файлы производится по тем же принципам – блоками. Для этого применяют процедуры:

BlockWrite (**var** F: **File**; **var** Buf; Count: Word; [**Var** Result: Word]); – пишет в открытый файл F Count блоков из переменной Buf. Параметр Result необязателен, содержит количество фактически обработанных записей.

BlockRead(**Var** F: **File**; **Var** Buf; Count: Word; [**Var** Result: Word]); – читает в переменную Buf из открытого файла F Count блоков.

Переменная Buf в данном случае может быть любого размера, типа и месторасположения, однако по размеру не меньше, чем <размер>*<размер_блока>, который был указан при открытии: процедуры не проверяют реальный размер переменных, а обрабатывают непрерывный участок памяти, начиная с указанной переменной. Желательно, чтобы размер записи, указанный при открытии файла, был кратен размеру блока, физически читаемого с диска, или размеру блока буфера MS-DOS. Позиционирование производится посредством `Seek` по блокам точно так же, как и для типизированных файлов.

Таким образом, программист имеет возможность обращаться к *любому* участку файла по своему усмотрению и в произвольном порядке. Кроме того, операции чтения/записи больших блоков выполняются быстрее, чем множество аналогичных операций для маленьких записей. Применение нетипизированных файлов очевидно: для сохранения/восстановления (динамически размещаемых) массивов данных переменного значимого размера.

ЧАСТЬ 4. КОМПОНЕНТЫ

Тема 12. Общие характеристики компонентов

12.1. ***Иерархия компонентов. Родительские и дочерние компоненты.***

Компонентами Delphi называются потомки класса TComponent. Фрагмент иерархии компонентов в Delphi показан на рис.12.1.

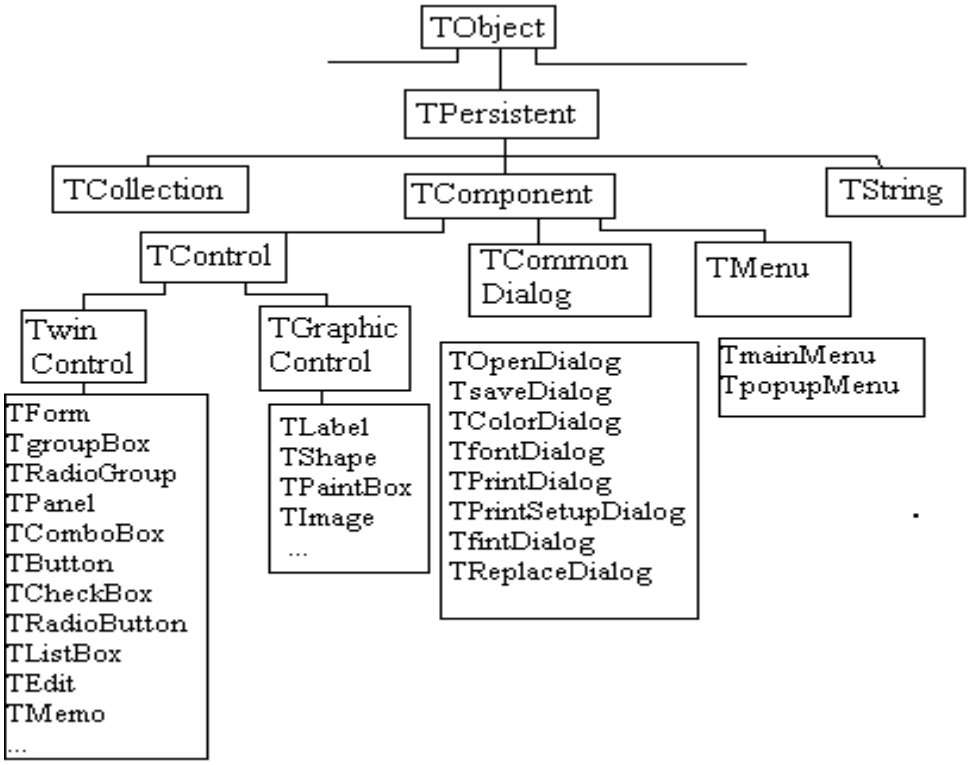


Рис. 12.1. Фрагменты иерархии компонентов в Delphi.

В соответствии с представленной иерархией все компоненты классифицируются:

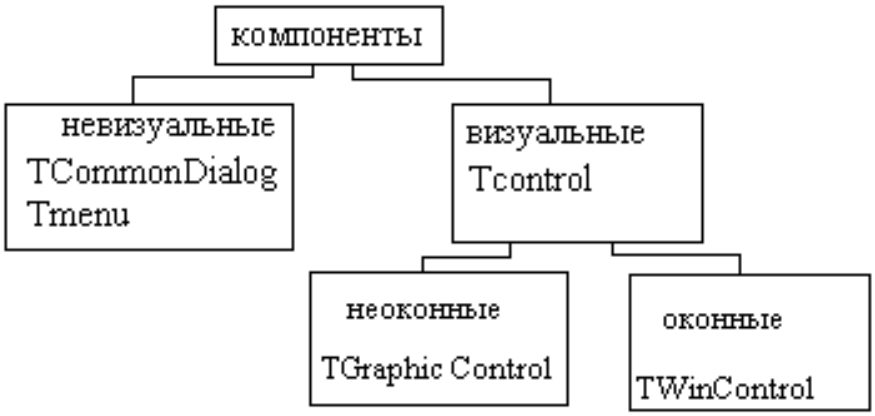


Рис. 12.2. Классификация компонентов.

Визуальный компонент в момент работы программы выглядит аналогично заданию в окне формы.

Невизуальный компонент не виден в момент работы программы, видно лишь создаваемое им окно (для компонентов диалога) или меню (для компонентов меню).

Оконные элементы управления:

- содержат дескриптор окна;
- могут содержать другие компоненты;
- могут становиться активными.

Для оконных компонентов можно устанавливать фокус ввода (SetFocus) и активизировать при помощи клавиши Tab.

Неоконные элементы управления:

- не содержат дескриптор окна;
- не могут содержать другие компоненты;
- не могут становиться активными.

Неоконные компоненты имеют свойство Canvas для задания графического рисунка (см. тему 14).

Оконные элементы управления могут содержать в себе другие компоненты. Элементы, содержащие другие компоненты, принято называть *родительскими*. Компоненты, расположенные внутри родительского компонента, – *дочерними*. Дочерние компоненты не могут выходить из границ своего родителя, они появляются и исчезают вместе с ним. Дочерними компонентами могут быть как оконные, так и неоконные компоненты. Например, компонент формы является родительским, а компоненты, расположенные в окне, – дочерними; компонент панель является родительским для расположенных внутри панели компонентов и дочерним для формы.

Для родительских /дочерних компонентов характерны свойства, которые будут рассмотрены в п. 12.2.

12.2. ***Общие свойства компонентов.***

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

Простые свойства – 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.

Перечислимые свойства – те, которые могут принимать значения из предопределенного набора (списка). Простейший пример – это свойство типа Boolean, которое может принимать значения True или False.

Вложенные свойства – те, которые поддерживают вложенные значения (или объекты). Object Inspector изображает знак "+" слева от названия этих 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. Некоторые свойства, – например, Font, – для изменения своих значений располагают возможностью вызвать диалоговое окно. Для этого достаточно щелкнуть маленькую кнопку с тремя точками в правой части строки Инспектора Объектов, показывающей данное свойство.

Общие свойства компонентов.

AutoSize: Boolean;	Разрешается/ запрещается компоненту автоматически изменять свои размеры в зависимости от размеров содержимого.
Aligment= (taLeftJustify, taRightJustify, taCenter);	Расположение текста относительно границ компонента: taLeftJustify – прижать к левой границе, taRightJustify – прижать к правой границе, taCenter – расположить по центру.
Caption: string;	Надпись на компоненте.
Color: Tcolor;	Цвет компонента.
Cursor: Tcursor;	Вид указателя мыши на компоненте.

Font: Tfont;	Шрифт.
Name: String;	Определение имени компонента.
Height: integer;	Высота.
Left:Integer;	Положение от левой кромки.
Top:Integer;	Положение от верхней кромки.
Width: Integer;	Ширина.
Visible: Boolean;	Видимость компонента. True – показывает, false – прячется.
Tag: integer;	Произвольный целочисленный параметр, который не используется в Delphi, и программист может распоряжаться им по своему усмотрению.

Свойства оконных компонентов.

Ctl3D: Boolean;	Позволяет создавать объемность изображения компонента.
Focused: Boolean;	Позволяет разместить на компоненте фокус ввода.
CanFocus: Boolean;	Разрешает/запрещает размещение фокуса ввода.
Enebled: Boolean;	Возможность активизации компонента. Если свойство имеет значение false, то компонент запрещен для выбора. Такие компоненты (точнее, надписи на них) обычно отображаются серым цветом.
TabStop: Boolean;	Разрешает/запрещает выбирать компонент клавишей Tab.
TabOrder: Integer;	Определяет порядок выбора элемента клавишей Tab.
Hint: String;	Подсказка к элементу.
ShowHint: Boolean;	Разрешает/запрещает показывать подсказку.
PopupMenu: TpopupMenu;	Определяет контекстное меню, появляющееся при нажатии правой кнопки мыши.

Свойства родительских компонентов, присущие только оконным компонентам:

Owner: Tcomponent;	Указывает на владельца компонента.
Controls[Index:integer]:Tcontrol;	Содержит список всех дочерних компонентов. Позволяет обращаться к конкретному дочернему компоненту по индексу.
ControlCount:integer;	Возвращает количество дочерних элементов.

Свойства дочерних компонентов (любой компонент может быть дочерним):

Align = (alNone, altop, alBottom, alleft, alright, alclient);	Выравнивание компонента относительно границ своего родителя: нет выравнивания (alNone), компонент прижимается к верхней границе (altop), нижней (alBottom), левой (alleft), правой (alright) , занимает всю свободную часть (alclient).
ParentColot: Boolean;	Использование/неиспользование родительского цвета при прорисовке компонента.
ParentCtl3d: Boolean;	Позволяет/не позволяет использовать такой же объем изображения, как у родительского компонента.
ParentFont: Boolean;	Использование/неиспользование родительского шрифта при задании надписи компонента.
ParentHint: Boolean;	Использование/неиспользование родительской подсказки для компонента.

*12.3. **События для компонентов.***

Основные действия пользователя при взаимодействии с приложением сводятся к работе с окном, перемещению мыши, нажатию кнопок мыши и нажатию клавиш на клавиатуре.

События для формы.

Oncreate	Возникает после создания, но появления компонента на экране.
OnActivite	Возникает после создания и при появлении компонента на экране.
OnClose	Закрытие окна

События, связанные с мышью.

OnClick	Щелчок мыши на компоненте.
OnDbClick	Двойной щелчок мыши на компоненте.
OnMouseDown	Нажатие клавиши мыши над компонентом.
OnMouseMove	Перемещение курсора мыши над компонентом.
OnMouseUp	Отпускание ранее нажатой кнопки мыши над компонентом.
OnStartDrag	Начало процесса «перетаскивания » объекта.

OnDragOver	Начало процесса «перетаскивания» объекта.
OnDragDrop	Отпускание ранее нажатой кнопки мыши после «перетаскивания» объекта.
OnEndDrag	Еще одно событие при отпускании ранее нажатой кнопки мыши после «перетаскивания» объекта.
OnEnter	Событие в момент получения элементом фокуса в результате нажатия мыши, клавиши табуляции или программной передачи фокуса. Только для оконных компонентов.
OnExit	Событие в момент потери элементом фокуса в результате манипуляции мышью, нажатия клавиши табуляции или программной передачи фокуса. Только для оконных компонентов.

Во все обработчики событий передается параметр **Sender: TObject**. Этот параметр содержит указатель на компонент, в котором произошло событие. Он не требуется, если пишется обработчик события для одного конкретного компонента. Однако часто один обработчик применяется для нескольких компонентов. При этом какие-то операции могут быть общими для любых источников событий, а какие-то требовать специфических действий. Если требуется распознать только тип объекта, можно использовать операцию **is**. Так, оператор

```
If (Sender is TListbox) then …
```

проверяет, не является ли источник события компонентом типа TlistBox.

Если требуется распознать объект по имени, можно использовать запись:

```
If (Sender = Edit1) then …
```

которая проверяет, не является ли имя источника события Edit1.

При событиях, связанных с мышью, передается еще ряд параметров. Заголовок обработчика события OnMouseDown может иметь, например, следующий вид:

```
Procedure Tform1.Edit1MouseDown (Sender:TObject; Button: TMouseButton;
                                 Shift:TShiftState; X,Y:integer);
```

Параметр Button типа TMouseButton = (mbRight, mbLeft, mbMiddle) определяет, какую кнопку мыши нажал пользователь.

Параметр Shift типа TshiftState = (ssShift, ssAlt, ssCtrl, ssRight, ssLEft, ssMiddle, ssDouble) определяет, какие вспомогательные клавиши были нажаты в момент нажатия мыши пользователем: Shift (ssShift), Alt (ssAlt), Ctrl (ssCtrl), правая кнопка мыши (ssRight), левая кнопка мыши (ssLEft), средняя кнопка мыши (ssMiddle), одновременно правая и левая кнопки мыши (ssDouble).

Параметр Button соответствует кнопке, нажимаемой в данный момент, а параметр Shift содержит информацию о том, какие кнопки были нажаты, включая и те, которые были нажаты раньше. Например, если пользователь нажмет левую кнопку мыши, а затем, не отпуская ее, нажмет правую, то после первого нажатия множество Shift будет равно [ssLeft], а после второго – [ssLeft, ssRight].

Параметры X,Y:integer передают координаты курсора в клиентскую область компонента.

События клавиатуры.

OnkeyDown	Событие наступает при нажатии пользователем любой клавиши. Можно распознать нажатые клавиши, включая функциональные, а также кнопки мыши, но нельзя распознать символ нажатой клавиши.
OnKeyPress	Событие наступает при нажатии пользователем клавиши символа. Можно распознать нажатую клавишу, но нельзя распознать функциональные клавиши.
OnKeyUp	Событие наступает при нажатии пользователем любой клавиши. Можно распознать нажатые клавиши, включая функциональные, а также кнопки мыши, но нельзя распознать символ отпускаемой клавиши.

Заголовок обработчика события OnKeyDown может иметь вид:

```
Procedure Tform1.Edit1KeyDown (Sender:TObject; var Key: Word; Shift:TShiftState);
```

Параметры Sender, Shift были рассмотрены выше. Параметр Key определяет код клавиши клавиатуры, нажатой в момент события. Параметр Key является целым числом типа Word и определяет не символ, а его код.

Приведем примеры кодов наиболее распространенных клавиш: Esc – 27, enter – 13, Shift – 16, Ctrl – 17, F1 – 112.

Например, реакцию на нажатие пользователем клавиши Enter можно оформить оператором:

```
If key = 13 then ....
```

Для определения символа по коду можно воспользоваться функциями ord или chr (см. пункт 8.2.):

Ord(символ) – возвращает код указанного символа.

Chr(код) – возвращает символ, соответствующий указанному коду.

Наример, если вы хотите распознать клавишу, соответствующую символу «Y», вы можете написать:

```
If key = ord('Y') then ....
```

12.4. ***Класс TStrings – набор строк.***

Абстрактный класс TStrings используется для работы с наборами строк. От него порождены потомки, обслуживающие наборы строк в таких компонентах как TComboBox, TListBox, TMemo и др.

Объекты TStrings обслуживают работу со строкой – объектом, поэтому строки можно сортировать, отыскивать нужный объект по описанию и т.д. Строки в наборе строки нумеруются (индексируются), начиная с нуля.

Свойства класса:

Commatext: string	Служит для получения всего набора строк в виде единой строки с кавычками и запятыми: «строка1», «строка2», «строка3», …
Count: Integer;	Количество строк в наборе.
Strings[index:integer]:String;	Открывает доступ к строке с индексом Index.
Text:string;	Служит для получения всего набора строк в виде единой длиной строки с разделителями EOLN: «строка1||строка2 ||строка3 ||строка4 ||…»

Методы класса:

Add(s:string):integer;	Добавляет строку в набор данных последней и возвращает ее индекс.
Append(s:string);	Добавляет строку в набор данных последней, но не возвращает ее индекс.
Clear;	Очищает набор данных.
Delete(Index:integer);	Удаляет строку с индексом Index.
Exchande(Index1, index2:integer;)	Меняет местами строки с index1 и index2.
IndexOf(S:string):integer;	Определяет индекс строки S в списке или возвращает –1, если такой строки нет.
Insert(index:integer; s;string);	Вставляет строку в набор с индексом Index.
Loadfromfile(FileName:string);	Открывает файл и считывает набор строк.
SaveToFile(file:string);	Сохраняет набор строк в файл.
Move(CurIndex,NwIndex:integer);	Перемещает строку из положения CurIndex в положение NewIndex.

*12.5. **Многооконное приложение. Диалоговые окна.***

Чаще всего сколько-нибудь сложное приложение не может ограничиться одним окном. Есть две различные модели приложений: с интерфейсом одного документа (SDI) и с интерфейсом множества документов (MDI).

Основным элементом любого приложения является – форма. Каждой новой форме, вводимой в приложение, соответствует свой модуль, описывающий эту форму как класс.

Свойство **BorderStyle** определяет общий вид окна и может принимать значения:

BsSizeable	Обычный вид окна с полосой заголовка, возможностью изменять размеры.
BsDialog	Неизменное по размерам окно, типичное окно диалогов.
BsSingle	Окно, размер которого пользователь не может изменить, потянув курсором мыши за край окна, но может изменять размеры кнопками в строке заголовка.
BsToolWindows	То же, что и BsSingle, но с меньшей полосой заголовка.
BsSizeToolWin	То же, что и BsSingle, но с меньшей по размеру полосой заголовка и отсутствием кнопок изменения размера в строке заголовка.
BsNone	Без полосы заголовка, не допускается изменение размеров и даже перемещения по экрану.

Свойство **BorderIcons** определяет набор кнопок, которые изменяются в полосе заголовка и принимает значения:

BySistemMenu	Кнопка системного меню.
ByMinimize	Кнопка *Свернуть*, сворачивает окно на панель задач.
ByMaximize	Кнопка *Развернуть*, разворачивает окно на весь экран.
ByHelp	Кнопка *Справки*.

Свойство **Icon** задает пиктограмму формы в заголовке окна. По умолчанию используется стандартная пиктограмма Delphi.

Свойство **formStyle** –основное свойство формы и может принимать значения:

FsNormal	Окно обычного приложения.
FsMDIForm	Родительская форма приложения, т.е. приложения с дочерними окнами, используемого при работе с несколькими документами.
FsMDIChild	Дочерняя форма приложения MDI.
FsStayOnTop	Окно, остающееся всегда поверх остальных окон.

При работе с несколькими окнами любое окно можно спрятать или показать с помощью методов:

Hide *– прячет компонент.*

Show *– показывает компонент.*

Close *– закрывает окно.*

BringToFont *– сделать верхним.*

SendToBack *– сделать нижним.*

При разработке многооконного приложения важно определить, какие окна будут появляться на экране при запуске приложения (AutoCreate Form), указать среди них главное окно Main form, которое будет находиться поверх созданных окон при запуске приложения, и перечислить окна, которые будут создаваться в процессе работы программы (Available Form).

Например, в созданном приложении три окна. При запуске приложения будут создаваться окна Form1, Form2, при этом окно Form1 – главное и будет

появляться верхним. Окно Form3 будет создаваться в процессе выполнения программы, по определению пользователя. Указать данные опции можно, выбрав команду Project/Option. При выборе пункта меню открывается диалоговое окно, в котором на страницу Forms определяются опции окон (см. рис. 12.3).

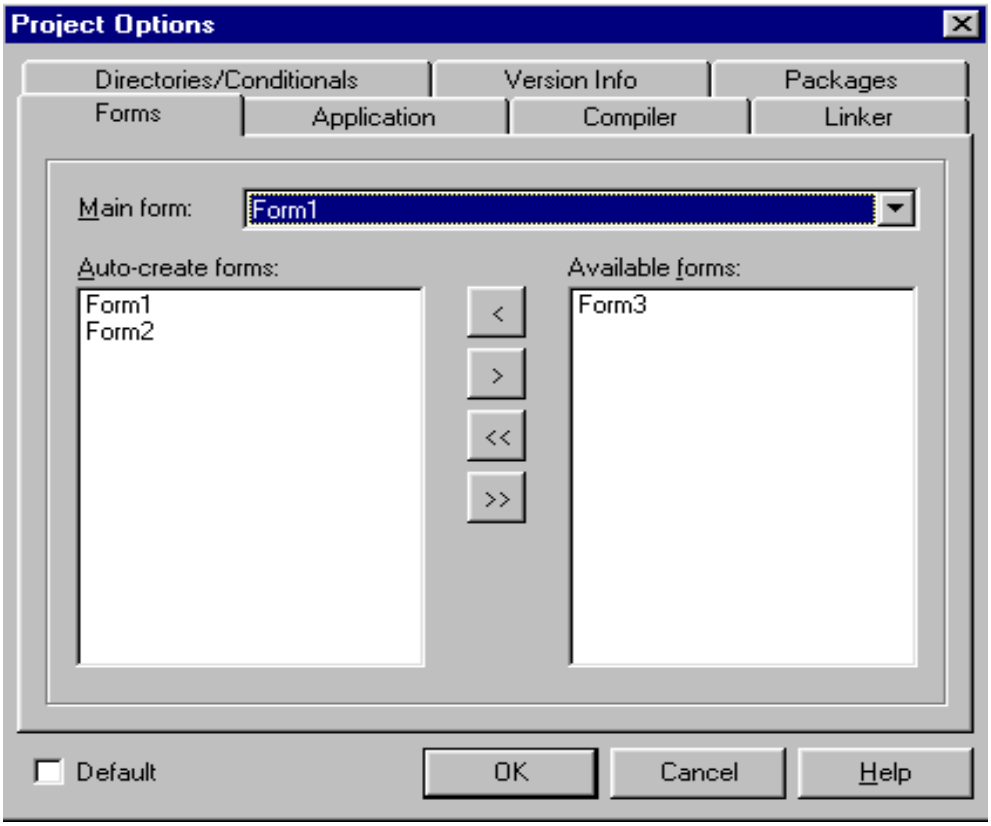


Рис. 12.3. Диалоговое окно «Опции проекта», страница «Формы».

В многооконных приложениях часто используют **модальные окна (окна сообщения) –** специальные окна, которые, появившись на экране, блокируют работу пользователя с другими окнами вплоть до своего закрытия. Обычно модальные окна используют для реализации диалога. Существуют несколько функций для вызова модальных диалоговых окон.

ShowMessage (S:String) – выводит окно сообщения.

Например, для вывода сообщения о разработчиках программы можно использовать команду:

```
ShowMessage('Программа разработана студентом 1 курса');
```

Результатом действия этой команды станет окно сообщения, показанное на рис. 12.4.

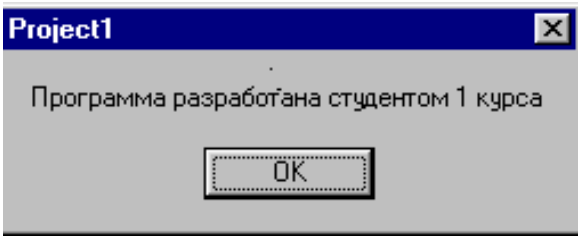


Рис.12.4. Пример окна, создаваемого функцией ShowMessage.

Следующая функция реализует работу окна сообщения со стандартным значком Windows и стандартными кнопками:

<результат>:= **MessageDlg**(<сообщение>,<тип>,<кнопки>,<контекст справки>);

Тип окна может принимать значения:

MtWarning	Внимание – черный восклицательный знак в желтом треугольнике.
MtError	Ошибка – белый крест в красном круге.
MtInformation	Информация – синяя латинская I на фоне белого облачка.
MtConfirmation	Запрос подтверждения – вопросительный знак на фоне белого облачка.
MtCustom	Пользовательское, обычное сообщение без значка.

Параметр кнопки может принимать значения: *mbNone, mbAbort, mbYes, mbOk, mbRetry, mbNo, mbCancel, mbIgnore, mbAll.*

Результатом функции является целое число и может принимать значения *mrNone, mrAbort, mrYes, mrOk, mrRetry, mrNo, mrCancel, mrIgnore, mrAll.*

Параметр <контекст справки> определяет номер экрана справочной системы, который появляется при нажатии на клавишу F1 в тот момент, когда окно сообщения находится на экране. Если использование справочной системы не предусмотрено, то при вызове функции в качестве параметра должен быть указан ноль.

Пример использования функции может быть следующий:

```
var rez:integer;
begin
        rez:=MessageDlg('Сохранить файл?', mt Confirmation,[mbYes,mbNo],0);
if rez=mrOk then …
        else …;
end;
```

Результатом действия этой команды будет окно сообщения, показанное на рис. 12.5.

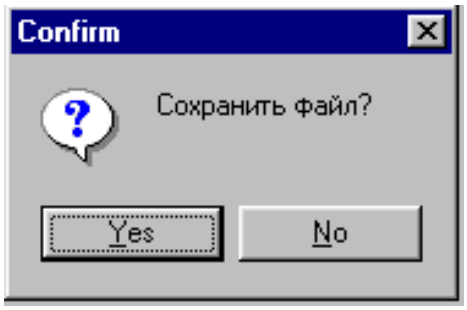


Рис.12.5. Пример окна, создаваемого функцией MessageDlg.

Диалоговое окно, которое позволяет пользователю ввести строку, создается функцией:

<результат>:= **InputBox** (<заголовок>,<сообщение>,<значение по умолчанию>);

Первоначально в строке редактирования записана строка по умолчанию, которая может быть изменена. Если пользователь выбирает кнопку Cancel, InputBox возвращает заданную по умолчанию строку. Если пользователь выбирает кнопку OK, функция InputBox в качестве результата возвращает строку в окне редактирования.

Пример использования функции может быть следующий:

```
var rez:String;
begin
rez:=InputBox('Добро пожаловать в программу', 'Как твоё имя?', 'студент');
…
end;
```

Результатом действия этой команды будет окно сообщения, показанное на рис. 12.6.

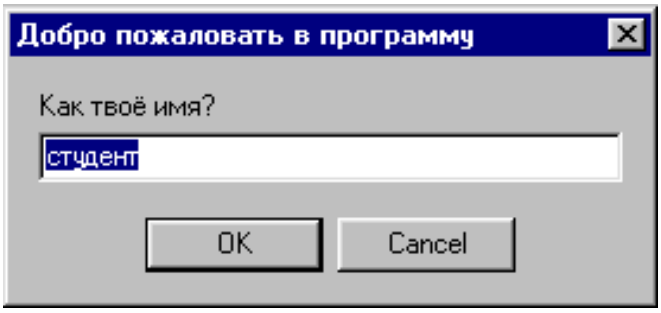


Рис.12.6. Пример окна, создаваемого функцией InputBox.

Если приложение должно знать, выбирает ли пользователь OK или Cancel, используется функция InputQuery.

<результат>:= **InputQuery**(<заголовок>,<сообщение>, <значение по умолчанию>);

Вид окна и параметры функции InputQuery аналогичны функции InputBox. Функция InputQuery возвращает тип Boolean. Если пользователь вводит строку в окно редактирования и выбирает OK, то параметр по умолчанию изменяется к новому введенному значению и результатом функции будет True. Если пользователь выбирает Cancel или нажимает клавишу Esc, то результатом функции будет false.

Тема 13. **Использование компонентов общего назначения**

Рассмотрим работу с некоторыми наиболее часто используемыми компонентами библиотеки VC.

Для размещения компонента на форме нужно щелкнуть мышью на компоненте, после щелкнуть на форме.

13.1. ***TFrame – рама.***

Компонент TFrame – рамка, контейнер для размещения других компонентов. В функциональном отношении компонент почти повторяет свойства формы и отличается от нее в основном лишь тем, что его можно помещать на формы или другие рамы. Для создания рамы TFrame нужно выбрать команду File/ New frame.

13.2. ***TLabel – метка.***

Компонент TLabel – метка служит для отображения текста на экране (рис. 13.1).

Рис.13.1. Пример размещения метки TLabel.

Метка предназначена для размещения на форме различного рода текстовых надписей. Для этого служит центральное свойство компонента – Caption. У компонента TLabel часто устанавливают свойство AutoSize =True, что позволяет метке увеличиваться и уменьшаться при изменении надписи. Если цвет метки совпадает с цветом фона, то при сокращении надписи до нулевой длины она может исчезнуть с экрана. В таком случае разыскать ее можно,

перейдя в окно Инспектора Объектов, развернув список компонент и выбрав ее из этого списка.

Свойство WordWrap: Boolean – разрешает/запрещает разрыв строки. Для вывода многострочных надписей задайте свойство AutoSize=false, WordWrap=True и установите подходящие размеры метки.

13.3. TEdit – однострочное окно ввода – вывода.

С помощью компонента TEdit можно вводить или отображать достаточно длинные строки (рис. 13.2).

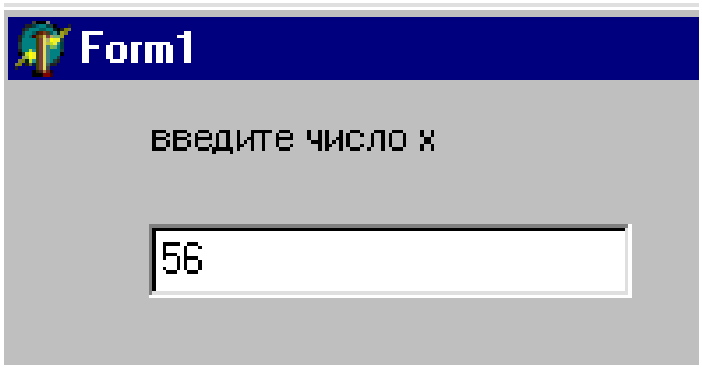


Рис. 13.2. Пример размещения на форме компонентов TLabel и TEdit.

Центральным свойством компонента является свойство Text, которое представляет собой отображаемую строку.

Свойства компонента:

AutoSelect: Boolean	Указывает, будет ли выделяться весь текст в момент получения компонентом фокуса ввода.
AutoSize: Boolean	Если содержит True и BorderStyle=bsSingle, высота компонента автоматически меняется при изменении высоты шрифта.
TborderStyle = bsNone, bsSingle;	Определяет стиль обрамления компонента: bsNone – нет обрамления, bsSingle – компонент обрамляется одной линией.
MaxLength: Integer;	Определяет максимальную длину текстовой строки. Если имеет значение 0, длина строки не ограничена.
Modified: Boolean;	Содержит True, если текст был изменен.
ReadOnly: Boolean;	Если содержит True, текст не может изменяться.
SelLength: Integer;	Содержит длину выделенной части текста.

SelStart: Integer;	Содержит номер первого символа выделенной части текста.
SelText: String;	Содержит выделенную часть текста.
Text: String;	Содержит весь текст.

Методы компонента:

Clear;	Удаляет весь текст.
ClearSelection;	Удаляет выделенный текст.
CopyToClipboard;	Копирует выделенный текст в буфер обмена ClipBoard.
CutToClipboard;	Копирует выделенный текст в буфер обмена ClipBoard, после чего удаляет выделенный текст.
SetFocus;	Устанавливает фокус ввода.
PastefromClipboard;	Вставляет текст из буфера обмена Clipboard.
SelectAll;	Выделяет весь текст.
Undo;	Восстанавливает текст в том виде, в каком он был перед последним получением компонентом фокуса ввода.

13.4. ***TMemo - многострочное окно ввода – вывода.***

Компоненты класса TMemo предназначены для ввода, редактирования и отображения достаточно большого текста (рис.13.3). Компонент представляет собой небольшой текстовый редактор, имеет ограничения на объем текста в 32Кб, что составляет 10-20 стр.

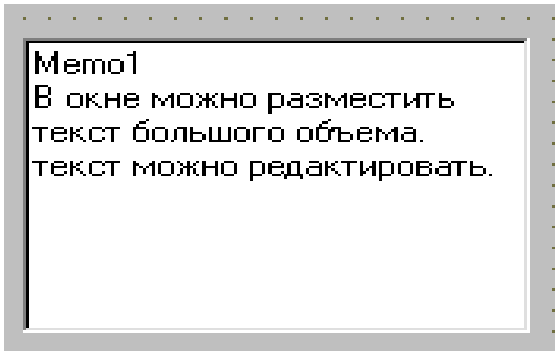


Рис.13.3. Компонент TMemo.

Текст хранится в свойстве Lines класса TStrings. В целом компонент представляет собой пронумерованный набор строк (нумерация начинается с нуля). С помощью свойств и методов класса TStrings (Count, Add, Delete,

Clear и т.д. – см. пункт 12.4 «Класс TStrings») можно динамически формировать содержимое компонента. Например:

```
edit1.text:=memo1.Lines.Strings[0];
memo1.Lines.Insert(2,'Иванов');
memo1.Lines.SaveToFile('c:\Мои документы\prim1.txt');
```

Компоненту присущи специфичные свойства:

Lines: Tstrings;	Содержит строки текста.
TscrollStyle = (ssNone, ssHorizontal, ssVertical, ssBoth)	Определяет наличие полос прокруток.
WantReturns: Boolean	Если содержит True, нажатие Enter вызывает переход на новую строку.
WantTabs: Boolean;	Если содержит True, нажатие Tab вызывает ввод в текст символа табуляции, который ставится нажатием Ctrl+Tab.
ReadOnly: Boolean;	Если содержит True, текст не может изменяться.
MaxLength: Integer;	Определяет максимальную длину текстовой строки. Если имеет значение 0, длина строки не ограничена.
Text: String;	Содержит весь текст.

13.5. ***TButton – командная кнопка.***

Кнопки TButton широко используются для управления программами (рис.13.4). TButton позволяет выполнить какие-либо действия при нажатии кнопки во время выполнения программы – событие OnClick.

Рис. 13.4. Компонент TButton.

В отличие от большинства других видимых компонентов кнопка Tbutton является компонентом самой Windows и не может изменять свой цвет произвольным образом – она его меняет вместе с изменением палитры Windows. У кнопки всегда системный цвет clBtnface и нет свойства Color. Шрифт

надписи на кнопке может менять свой стиль и размер, но компонент игнорирует изменение его цвета.

13.6. ***TScrollBar – полоса прокрутки.***

Полоса прокрутки, управляющий элемент, представляет собой ось координат. Вдоль оси координат перемещается ползунок. Для полосы прокрутки характерны свойства:

Kind = (sbhorizontal, sbVertical);	Размещает полосу горизонтально/вертикально.
Min: Integer;	Задает начальную координату оси.
Max: Integer;	Задает конечную координату оси.
Position: Integer;	Устанавливает ползунок в указанную позицию.
SmallChange	Малый сдвиг ползунка при щелчке мыши по концевой кнопке.
LargeChange	Большой сдвиг перемещения ползунка при щелчке мыши рядом с концевой кнопкой.

13.7. ***TCheckBox – флажок.***

Флажок TCheckBox – независимый переключатель используется для того, чтобы пользователь мог указать свое решение типа *ДА/ НЕТ/ НЕ ЗНАЮ (*в последнем случае в окошке компонента устанавливается флаг выбора, но само окошко закрашивается серым цветом*)*.

Рис. 13.5. Компонент TCheckBox.

Свойства компонента:

Alignment = (taLeftJustify, taRightJustify);	Определяет положение текста с левой/ правой стороны компонента.
State = (cbUnchecked, cbChecked, cbgrayed);	Содержит состояние компонента cbUnchecked - нет, cbChecked - да, cbgrayed – не знаю.
AllowGrayed: Boolean;	Разрешает/ запрещает использование состояния cbGrayed *(Не знаю).*

Caption: String;	Содержит связанный с компонентом текст.
Checked: Boolean;	Содержит выбор пользователя True - ДА (cbChecked), False - НЕТ(cbUnchecked cbUnchecked, cbGrayed).
State = (cbUnchecked, cbChecked, cbgrayed);	Содержит состояние компонента cbUnchecked – нет, cbChecked – да, cbGrayed – не знаю.

Для использования компонента с возможностью выбора только ДА/НЕТ нужно свойство AllowGrayed = false (запрет значения НЕ ЗНАЮ), и в окне кода программы чаще всего имеется запись:

```
If checkBox1.Checked
   then .....
   else .....;
```

В случае возможности выбора ДА/НЕТ/НЕ ЗНАЮ в окне кода программы используется типичная запись:

```
case CheckBox1.State of
  cbChecked: .......;
  cbunchecked: ......;
  cbGrayed: .........;
end;
```

Состояние флажка не зависит от состояния остальных флажков, поэтому такие переключатели называются *независимыми*. Свойство *Color* – компонента фактически игнорируется, а связанный с переключателем текст указывается в свойстве *Caption*.

13.8. ***TRadioButton – радиокнопка.***

В отличие от TCheckBox компоненты TRadioButton представляют собой зависимые переключатели, предназначенные для выбора одного из нескольких решений. На форму (точнее, на панель) помещаются по меньшей мере

два таких компонента. Когда включена одна радиокнопка, остальные выключены.

Радиокнопки могут иметь только два состояния: ДА/НЕТ, определяемые свойством Checked. Радиокнопки также имеют свойства *Caption, Alignment*. Как и в TCheckBox, невозможно изменять размеры и цвет круглого окошка компонента.

13.9. ***TRadioGroup – группа радиокнопок.***

Компонент класса TRadioGroup представляет собой специальный контейнер для размещения радиокнопок TRadioButton (рис.13.6).

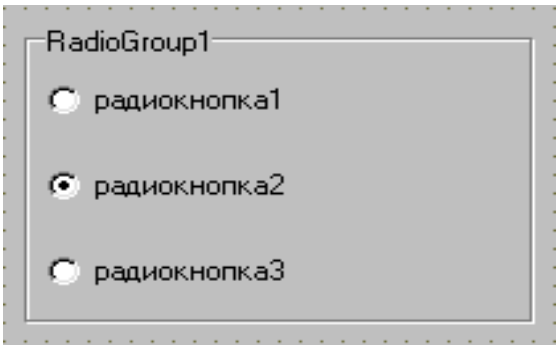


Рис. 13.6. Группа радиокнопок TRadioGroup.

Каждый размещаемый на контейнере переключатель (радиокнопка) помещается в специальный список Items: TStrings. В данном списке каждая радиокнопка пронумерована, начиная с нуля. Работа со списком радиокнопок осуществляется аналогично работе с классом TStrings (см. пункт 12.4).

Свойства компонента:

Columns: Integer;	Определяет количество столбцов переключателей.
Items: TStrings;	Содержит список строк с заголовками радиокнопок. Добавление/удаление элементов достигается согласно строкам списка (см. п. 12.4).
ItemIndex: integer;	Содержит индекс включенной радиокнопки. Если индекс равен –1, то все радиокнопки выключены.

Типичная работа со списком радиокнопок:

```
case radiogroup1.ItemIndex of
   0: .....;          {выбран 0 переключатель}
   1: ......;         {выбран 1 переключатель}
   2: ......;         {выбран 2 переключатель}
   .........
 else ......          {не выбран ни один переключатель}
end;
```

13.10. ***TGroupBox – панель группирования.***

Этот компонент служит контейнером для размещения дочерних компонентов и представляет собой прямоугольное окно с рамкой и текстом в верхней части (рис. 13.7).

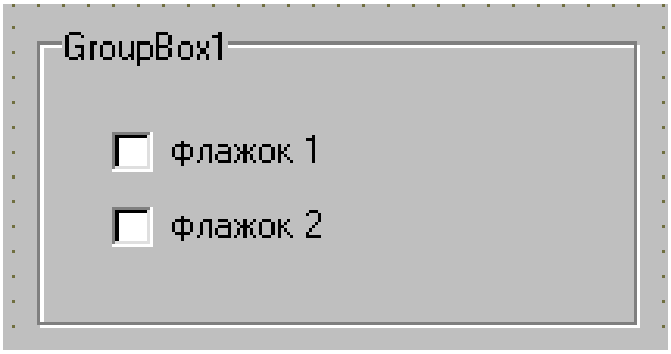


Рис. 13.7. Панель группирования.

Обычно с помощью компонента группируют флажки и другие управляющие элементы, объединенные по смыслу. Заголовок панели указывается свойством Caption.

13.11. ***TPanel – панель.***

Компонент TPanel представляет собой контейнер общего назначения. В отличие от TGroupBox он не имеет заголовок, поэтому менее удобен для группирования. Свойство Caption отображается в виде текстовой строки час-

то по центру, поэтому компонент удобно использовать для вывода сообщения (рис.13.8).

Рис. 13.8. Компонент TPanel.

Компонент имеет средства создания различных эффектов трехмерности за счет использующихся в нем двух кромок – внешней и внутренней.

Свойства компонента:

BevelOuter = (bvRaised, bvLowered, bvNone);	Параметры внешней рамки: выпуклая, вдавленная, отсутствует.
BevelInner = (bvRaised, bvLowered, bvNone);	Параметры внутренней рамки: выпуклая, вдавленная, отсутствует.
BewelWidth: Integer;	Ширина рамок.
BorderWidth: Integer;	Ширина бордюра (состоит из двух рамок).
Caption: String;	Задание надписи.

13.12. ***TListBox – список выбора.***

Компонент класса TListBox представляет собой стандартный список выбора, с помощью которого пользователь может выбрать один или несколько элементов (рис.13.9). В компоненте предусмотрена возможность программной прорисовки элементов, поэтому список может содержать не только строки, но и произвольные изображения.

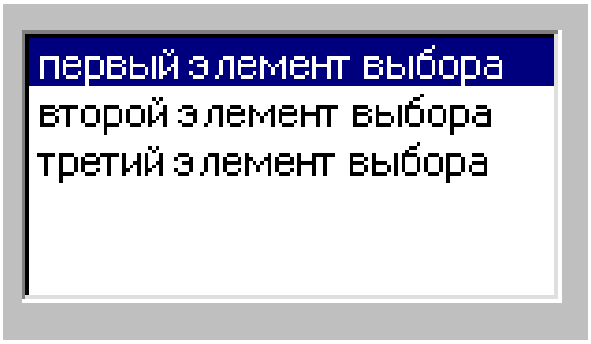


Рис. 13.9. Компонент TListBox.

Свойства компонента:

Canvas: Tcanvas;	Канва для программной прорисовки элементов.
Columns: Longint;	Определяет количество колонок элементов в списке.
ItemHeight: integer;	Определяет высоту элемента в пикселях.
ItemIndex: Integer;	Содержит индекс выбранного (выделенного) элемента.
Items: Tsrings;	Содержит набор строк элементов.
MultiSelect: Bollean;	Разрешает/отменяет выбор нескольких элементов.
SelCount: Integer;	Содержит количество выбранных элементов.
Sorted: Boolean;	Разрешает/отменяет сортировку строк в алфавитном порядке.

Элементы списка задаются в списке Items:TStrings, нумеруются с нуля. Создание элементов списка компонента реализуется с помощью методов свойства Items – Add, Append, Insert и других методов класса TStrings (см. пункт 12.4).

13.13. ***TComboBox - раскрывающийся список выбора.***

Комбинированный список TComboBox представляет собой комбинацию списка TListBox и редактора TEdit, поэтому большинство его свойств и методов заимствовано у этих компонентов (рис.13.10).

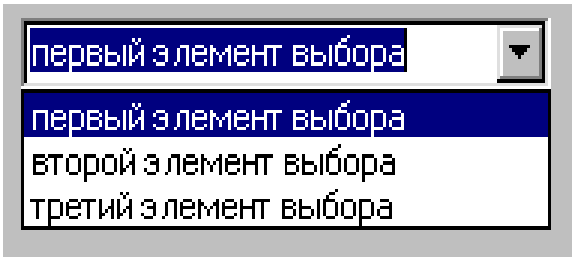


Рис. 13.10. Компонент TComboBox.

Существует пять модификаций компонента, определяемые свойством Style:

CsSimple – список всегда раскрыт;

CsDropDown – список раскрывается при нажатии на кнопку справа от редактора;

CsDropDownList – режим отображения выбора, и его нельзя использовать для отображения новой строки;

CsOwerDrawFixed - для программной прорисовки списка;

csOwerDrawvariable – для программной прорисовки списка.

Верхняя часть TComboBox подобна редактору TEdit и имеет почти все свойства TEdit. Обращение к строке, написанной в окне редактора, будет выглядеть: *ComboBox1.text.*

Нижняя часть TComboBox подобна списку выбора TListBox и имеет соответствующие свойства и методы. Например, для обращения ко второму элементу списка используется запись: *ComboBox1.Items.Strings(2).*

Для обращения ко выделенному элементы списка можно написать: *ComboBox1.Items.Strings(Combobox1.itemindex).*

Работа со списком ведется методами Add, Insert, delete и т.п. (см. п.12.4).

Специфичные события и методы компонента:

DropDownCount: Integer;	Определяет количество элементов списка, при котором еще не будет появляться полоса прокрутки списка.
DroppedDown: Boolean;	Определяет, раскрыт ли список в данный момент.
OnDropDown:TnotifyEvent;	Событие происходит при изменении состояния списка.

13.14. ***TMainmenu – главное меню.***

Компонент класса TMainMenu определяет главное (горизонтальное) меню формы. Невизуальный компонент. Это компонент списка и для задания пунктов меню используется свойство Items. Свойство Items открывает конструктор меню, в котором выделяя элемент списка (пункт меню) с помощью свойства Caption задается надпись каждого пункта (рис.13.11). Для вставки разделительной черты, отделяющей группы пунктов, назовите очередной пункт именем «-». Для создания разветвленных меню (пункты вызывают вспомогательные списки), щелкните по пункту и нажмите Ctrl+Вправо.

Рис. 13.11. Конструктор компонента TMainMenu.

Основные свойства:

Bitmap: Tbitmap;	Позволяет разместить возле надписи на пункте меню рисунок.
Checked: Boolean;	Если True, рядом с пунктом появляется галочка.
Items [Index:integer]: TmenuItem;	Обращение к любому пункту меню.
MenuIndex: Integer;	Определяет индекс данного пункта в родительском компоненте.
RadioItem: Boolean;	Определяет работу с пунктом на подобии радиокнопки.

Каждый элемент меню является объектом класса TMenuItem. Основным событием для каждого пункта меню OnClick, которое возникает при щелчке мыши на пункте или при нажатии кнопки Enter.

*13.15. **TPopupMenu – контекстное меню.***

Компонент TPopupMenu используют для создания вспомогательного меню, появляющегося после нажатия правой кнопки мыши. Контекстное меню может быть создано для любого оконного компонента. Чтобы связать контекстное меню с оконным компонентом, необходимо для оконного компонента свойство PopupMenu установить True.

Вспомогательное меню создается с помощью конструктора меню и содержит пункты меню только в один столбец. Элементы контекстного меню имеют тип TMenuItem и создаются аналогично главному.

13.16. ***TSringGrid - таблица строк.***

Компонент TStringGrid предназначен для создания таблиц, в ячейках которых располагаются произвольные текстовые строки (рис.13.12). Компонент находится на странице Additional .

Таблица делится на две части – фиксированную и рабочую. Фиксированная служит для показа заголовков строк/столбцов и выделена цветом, рабочая – это остальная часть таблицы. Они могут содержать произвольное количество строк и столбцов. Если рабочая часть целиком не уменьшается в пределах окна компонента, в этом случае автоматически появляются полосы прокрутки.

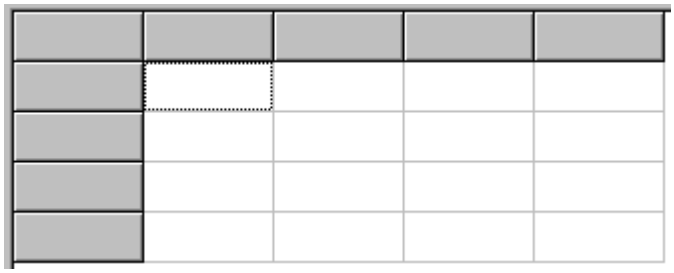


Рис.13.12. Таблица строк компонента TStringGrid.

Фиксированная часть таблицы определяется свойствами:

FixedCols: integer – количество фиксированных столбцов;

FixedRows: integer – количество фиксированных строк.

Если эти свойства равны нулю, то таблица не содержит фиксированной части.

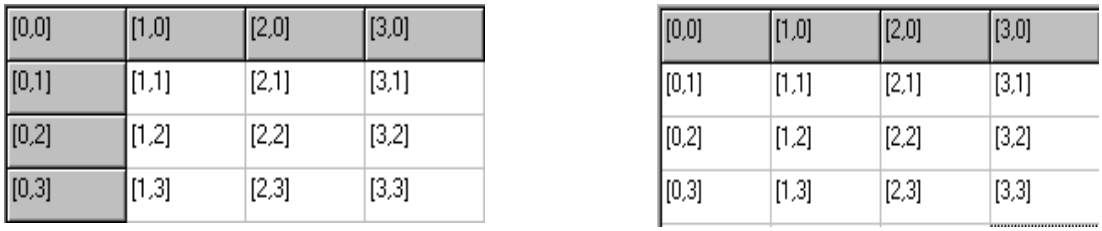
Размер рабочей части определяется свойствами:

ColCount: integer – количество столбцов;

RowCount: integer – количество строк.

Центральным свойством компонента является свойство *Cells: string* – двумерный массив ячеек. Ячейки находятся на пересечении столбца и строки. В массиве строки и столбцы нумеруются, начиная с нуля. Массив ячеек включает фиксированную и рабочую часть. На рис.13.13 представлены примеры нумерации ячеек в массиве: а) в массиве с одной фиксированной стро-

кой и одним фиксированным столбцом, б) в массиве с одной фиксированной строкой.



А *Б*

Рис. 13.13. Примеры нумерации ячеек в массиве: а) с одной фиксированной строкой и одним фиксированным столбцом, б) с одной фиксированной строкой.

К каждой ячейки можно обратиться:

Cells[<номер столбца>, <номер строки>]

В ячейки можно записывать информацию или читать содержимое ячеек. Например, записать информацию в ячейку можно следующим образом:

StringGrid1.cells[0,0] := 'это верхняя левая ячейка ';

Свойства компонента:

BorderStyle =(bsSingle, bsNone);	Определяет рамку компонента.
Cells(Acol, Arow: integer);	Обращение к ячейки с координатами Acol, Arow.
Col: Longint;	Содержит номер столбца сфокусированной ячейки.
ColCount: Longint;	Количество столбцов таблицы.
Cols[Index: Integer]: Tstrings;	Содержит весь столбец с индексом Index.
ColWidths[Index: Integer]:Tsrings;	Содержит ширину столбца с индексом Index.
DefaultColWidth:Integer;	Значение по умолчанию ширины столбца.
DefaultRowHeight: integer;	Значение по умолчанию высоты строки.
EditorMode: Boolean;	Разрешает/ запрещает редактирование ячеек. Игнорируется, если выключено goEditing.
FixedColor: Tcolor;	Определяет цвет фиксированной зоны.
FixedCols: Integer;	Количество столбцов фиксированной зоны.
FixedRows: Integer;	Количество строк фиксированной зоны.
GridHeight: integer;	Высота таблицы.

GridLineWidht: Integer;	Толщина линий, расчерчивающих таблицу.
GridWidht:Integer;	Ширина таблицы.
Options	Параметры таблицы.
Row: Longint;	Содержит номер строки сфокусированной ячейки.
RowCount: Longint;	Количество строк таблицы.
RowHeights[Index: Longint]: Integer;	Содержит высоту строки с индексом Index.
Rows[Index: integer]: Tstrings;	Содержит всю строку с индексом Index.

При запуске программы содержимое ячеек можно редактировать. Для этого нужно задать свойство *Option* – параметры. В инспекторе объектов слева от свойства Option стоит знак "+", значит, свойство вложенное и щелчок мыши на знаке плюс, открывает множество значений свойства.

Перечислим некоторые элементы множества Option:

GoFixed-VertLine	Столбцы фиксированной зоны разделяются вертикальными линиями.
GoFixedHorzLine	Строки фиксированной зоны разделяются горизонтальными линиями.
GoVertLine	Столбцы рабочей зоны разделяются вертикальными линиями.
GoHorzLine	Строки рабочей зоны разделяются горизонтальными линиями.
GoRangeSelect	Разрешено выделение нескольких ячеек. Игнорируется, если включен элемент goEdit.
GoRowSizing	Разрешено ручное (мышью) изменение высоты строк.
GoColSizing	Разрешено ручное (мышью) изменение ширины столбцов.
GoRowMoving	Разрешено ручное (при помощи мыши) перемещение строк.
GoColMoving	Разрешено ручное перемещение столбцов.
GoEditing	Разрешено редактирование ячеек. Игнорируется, если включен элемент goRowSelect.
GoTabs	Разрешено обходить ячейки клавишей Tab.
GoRowSelect	Обязывает выделять сразу все ячейки строки и запрещает редактирование ячеек.

13.17. ***TDrawGrid – произвольная таблица.***

Компонент TDrawGrid представляет программисту мощные возможности создания и обслуживания табличных структур данных. Он обеспечивает двумерное представление данных. Компонент находится на странице Additional.

Компонент TDrawGrid является непосредственным родителем строковой таблицы TStringGrid, поэтому имеет все свойства компонента TStringGrid, за исключением специфичных для строк свойств Cells, Cols, Rows.

Для прорисовки используется свойство Canvas. Для прорисовки той или иной ячейки используется событие OnDrawCell.

Для компонент TStringGrid и TDrawGrid используются методы:

CellRect (Acol:integer, Arow:integer);	Возвращает прямоугольник ячейки по номерам столбца Acol и строки Arow.
MoveTocell (X,Y:integer; var Acol, Arow:Longint);	Возвращает табличные координаты ячейки Acol, Arow по экранным координатам (X,Y) точки.

13.8. ***TBitBtn – командная кнопка с изображением.***

Кнопка TBitBtn представляет собой компонент TButton с изображением, рисуемом на поверхности кнопки (рис.13.14). Компонент находится на странице Additional .

Рис.13.14. Примеры кнопок TBitBtn.

Свойства компонента:

Caption: String;	
Kind = (bkOK, bkCancel, bkHelp, bkYes, bkNo, bkClose, …);	Определяет стандартные кнопки: ДА, НЕТ, ОТМЕНА, и др.
Glyph:Tbitmap;	Определяет связанное с кнопкой изображение.

При выборе свойства Glyph открывается редактор картинок (рис.13.15). Выбрав кнопку LOAD, можно указать графический файл с нужной картинкой, которая будет размещена на компоненте.

Рис. 13.15. Окно Picture Editor, открывающиеся при задании свойства Glyph.

13.19. ***TUpDown – спаренная кнопка.***

Компонент TUpDown предназначен для регулирования числовой величины (рис.13.16). Он имеет пару кнопок, с помощью которых величина увеличивается или уменьшается.

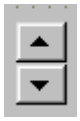


Рис. 13.16. Компонент TUpDown.

Обычно компонент TUpDown связан с компаньоном – другим компонентом класса TEdit, который отображает регулируемую величину и при необходимости может ее редактировать. Для связи компонента с другим компонентом класса TEdit существует свойство Associate. Например, на форме располагается компонент Edit1 и рядом компонент UpDown1 (рис.13.17). Для связи компонента UpDown1 с компонентом Edit1 свойство Associate=Edit1.

Рис 13.17. Связанные компоненты TEdit и TUpDown.

Свойства компонента:

TUDAlignButton = (udLeft, udRight);	Определяет положение компонента относительно компаньона: слева, справа.
Associate: TwinControl;	Определяет компаньона – связанный компонент.
Increment: smallint;	Шаг изменения величины.
Max: smallint;	Максимальное значение величины.
Min: smallint;	Минимальное значение величины.
Position: smallint;	Текущее значение величины.
Wrap: Boolean;	Разрешает/запрещает выход значений из диапазона [min; max].

13.20. ***TOleContainer – контейнер объектов OLE.***

Компонент является удобным контейнером для размещения связанного или внедренного OLE – объекта. Замечательной особенностью OLE – объекта является то, что его активизация с помощью двойного щелчка мыши приводит к активизации связанной с объектом программы.

Для вставки OLE-объекта необходимо на форме разместить компонент класса TOleContainer, вызвать для компонента контекстное меню (щелкнуть правой мышью на компоненте) и выбрать команду Insert Object. После выбора команды открывается диалоговое окно «Вставка объекта» для создания или открытия файла любого приложения (рис.13.18).

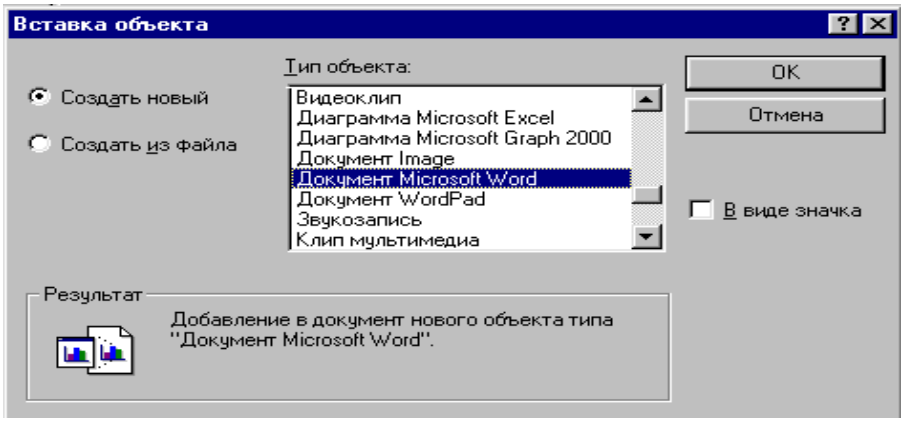


Рис.13.18. Диалоговое окно выбора вставляемого объекта в компонент TOLECntainer.

Свойство Align позволяет выравнивать OLE-компонент на форме или другом контейнере.

Тема 14. **Графический инструментарий**

В Delphi созданы специальные классы-надстройки, созданные для использования в графических инструментах Windows: для контекста – класс TCanvas, для шрифта – TFont, для пера – TPen, для кисти – TBrush.

14.1. ***Класс TFont.***

С помощью класса TFont создаются объект-шрифт для любого графического устройства (в частности, объекты на экране).

Для данного класса характерны свойства:

Color: TColor;	Цвет шрифта.
Height: Integer;	Высота шрифта в пикселях экрана.
Name: TfontName;	Имя шрифта.
Size: Integer;	Высота шрифта в пунктах (1/72 дюйма).
Style = [fsbold, fsItalic, fsUnderLine, fsStrike-Out]	Стиль шрифта. Может принимать комбинацию элементов: fsbold (жирный), fsItalic (курсив), fsUnderLine (подчеркнутый), fsStrikeOut (перечеркнутый).

Почти каждый компонент имеет свойство Font:TFont. Слева от свойства стоит знак «+», который указывает, что свойство вложенное. Щелчок мышью на знаке плюс открывает вложенные свойства класса Tfont.

14.2. ***Классы TPen, TBrush.***

С помощью класса **TPen (Перо)** создается объект Перо, служащий для вычерчивания линий, контуров графических рисунков.

Свойства класса:

Color: TColor;	Цвет вычерчиваемых пером линий.
Style: TPenStyle;	Определяет стиль линий: сплошная, пунктирная и т.д.
Width: Integer;	Толщина линий в пикселях экрана.
Mode: TpenMode;	Определяет способ взаимодействия линий с фоном.

Объект класса **TBrush (Кисть)** служит для заполнения внутреннего пространства замкнутых фигур.

Свойства класса:

Bitmap: TBitMap;	Содержит растовое изображение, которое будет использоваться кистью для заполнения. Если это свойство определено, то свойства Color, Style игнорируются.
Color: TColor;	Цвет вычерчиваемых пером линий.
Style: TBrushStyle;	Определяет стиль линий заполнения: сплошная закраска, диагональные линии, горизонтальные линии и т.д.

14.3. ***Класс TCanvas.***

Данный класс создает «канву», на которой можно рисовать. Объекты класса TCanvas автоматически создаются для всех видимых неоконых компонентов.

Класс имеет свойства Pen: TPen, Brush: TBrush, Font: TFont.

Задание цвета пера и кисти можно осуществить несколькими способами. Например, непосредственно указав стандартные цвета Windows (clred, clblue, clgreen и т.д.). Наиболее распространенным способом задания цвета в Windows является цветовая модель RGB (по первым буквам названий цветов: Red – красный, Green – зеленный, Blue – голубой). Комбинируя эти три цвета, можно получить любой цвет. Для этого интенсивность каждого из трех составляющих цветов задается значением от 0 до 255.

Рассмотрим основные свойства и методы класса TCanvas.

PIXELS[x,y: Integer]: TColor – это двумерный массив, который отвечает за цвет пикселя. Координаты пикселя в массиве определяются согласно рис. 14.1.

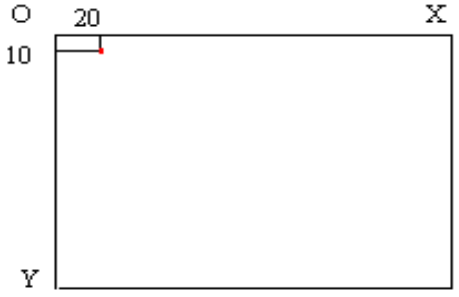


Рис. 14.1. Координатная сетка в массиве пикселей.

Например, для размещения красной точки в объекте Label1 с координатами (10,20) следуют записать:

```
Label1.Canvas.Pixil(10,20):=clred;
```

PenPos:Tpoint – определяет текущее положение пера в пикселях относительно левого верхнего угла канвы.

MoveTo(X,Y: integer) – перемещение пера, курсор устанавливается, не рисуя, в позицию (X,У).

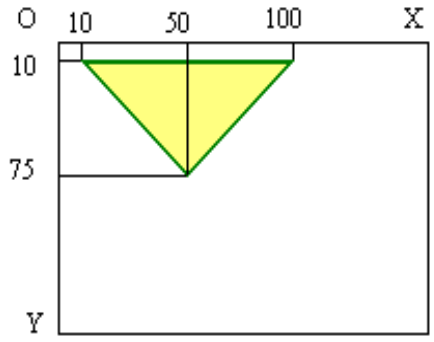
LineTo(X,Y) – рисуется отрезок от позиции, в которой установлено перо, до позиции (X,У).

PolyLine(Points: array of TPoints) – рисование контура многоугольника. Для этого используется массив POINT:TPOINT, передающий последовательность точек, которые функция PolyLine соединяет линиями. Чтобы многоугольник получился замкнутым, необходимо соединить позиции последней и первой точек. Пример рисования треугольника:

```
label1.Canvas.Pen.Color:=clgreen;
Label1.Canvas.PolyLine (Points(10,10), Points(50,75), Points(100,10), Points(10,10));
```

Polygon (Points: array of TPoints) – рисование закрашенных многоугольников. Контур рисуется установленными свойствами пера, при закрашивании используются текущие свойства кисти. Пример рисования закрашенного треугольника:

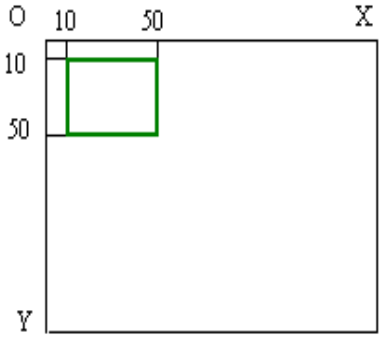
```
Label1.Canvas.Pen.Color := clgreen;
Label1.Canvas.Pen.Width := 4;
label1.Canvas.Brush.Color:=clYellow;
Label1.Canvas.Polygon (Points(10,10),
Points(50,75), Points(100,10),
Points(10,10));
```



Rectangle(x1,y1,x2,y2: integer) – рисование контура прямоугольника. Рисуется контур прямоугольника, где (x1,y1) – координаты верхнего левого угла, (x2,y2) – координаты нижнего правого угла. При рисовании использу-

ются текущие свойства пера, которые можно задать перед применением процедуры рисования. Например:

```
Label1.Canvas.Pen.Color := clgreen;
Label1.Canvas.Pen.Width := 4;
Label1.Canvas.Rectangle(10,10,50.50);
```

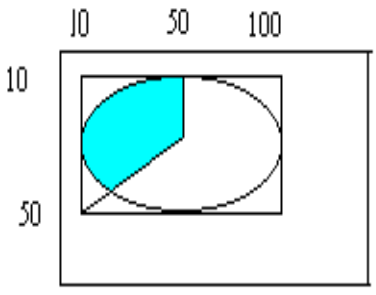


FillRect(Rect(x1,y1,x2,y2:integer)) – рисуется закрашенный прямоугольник, где функция Rect(x1,y1,x2,y2) возвращает размер прямоугольной области аналогично функции Rectangle, в которой (x1,y1) – координаты верхнего левого угла, (x2,y2) – координаты нижнего правого угла. Контур рисуется установленными свойствами пера, при закрашивании используются текущие свойства кисти.

Ellipse(X1,Y1,X2,Y2: integer) – рисование закрашенных окружностей и эллипсов. Чертит эллипс в охватывающей прямоугольной области. При рисовании окружностей указываются размеры квадратной области.

Pie(x1,y1,x2,y2,x3,y3,x4,y4:integer) – рисование секторов эллипса. При этом первые четыре параметра указывают размер прямоугольной области. Параметры X3,У3,X4,У4 определяют часть эллипса, которая будет показана. Начало дуги лежит на пересечении эллипса и луча, проведенного из его центра в точку (x3,y3), а конец – на пересечении с лучом из центра в точку (x4,y4). Дуга чертится против часовой стрелки. Например:

```
Label1.Brush.Color := clblue;
Label1.Canvas.Pie (10,10,100,50,50,10,10,50);
```



TextOut(x,y:integer; S:string) – выводит текстовую строку S, начиная с позиции (x,y). Перед применением функции можно задать параметры шрифта. Например:

```
Label1.canvas.font.Style:=[fsbold];
Label1.canvas.TextOut (1,1,'чертеж');
```

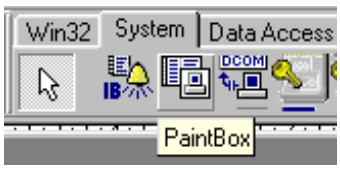
14.4. ***Компонент TShape, TImage, TPaintBox, TChart.***

 **TShape** – стандартная фигура. Компонент рисует одну из простейших геометрических фигур: прямоугольник, квадрат, скругленный прямоугольник, скругленный квадрат, эллипс, окружность.

Свойства компонента:

Shape = (stRectangle, stSquare, stRoundRect, stRoundSquare, stEllipse, stCircle);	Определяет вид фигуры: stRectangle – прямоугольник, stCircle – окружность, stSquare – квадрат, stEllipse – эллипс, stRoundRect – скругленный прямоугольник, stRoundSquare – скругленный квадрат.
+ Brush: Tbrush;	Кисть, служит для заполнения внутреннего пространства фигур. Имеет вложенные свойства (см. пункт 14.2).
+ Pen: Tpen;	Перо, служит для вычерчивания линий, контуров графических рисунков.

 **TImage** – отображение картинок. Этот компонент служит для размещения на форме трех видов изображения: растровой картинки, пиктограммы или метафайла. Изображение содержится в свойстве компонента *Picture*. Свойство *Canvas* позволяет отредактировать растровое изображение.



TPaintBox – окно для рисования. Компонент размещает на форме прямоугольную область (канву) для рисования. Если при рисовании фигуры ее размеры окажутся больше компоненты TPaintBox, то

фигура окажется усеченной – не выйдет за пределы области рисования. Создав обработчик события OnPaint для TPaintBox, можно передвигать изображение по форме, изменяя у TPaintBox свойства Left и Top. TPaintBox использует свойство ALIGH для удержания изображения сверху, снизу, сверху, внизу формы или заполнять всю область формы.

TChart – построитель графиков. Компонент предназначен для графического представления числовых данных (рис.14.2).

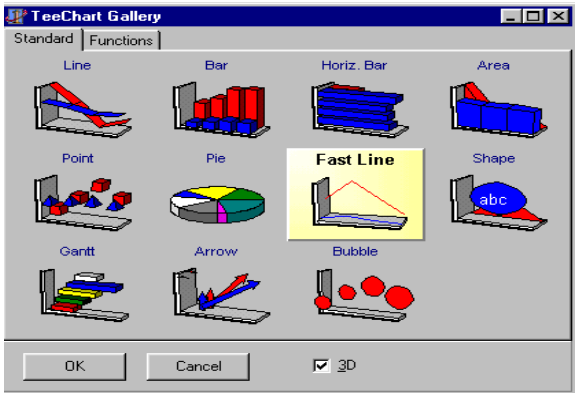


Рис. 14.2. Возможные виды графиков компонента TChart.

После размещения компонента на форме нужно щелкнуть по нему правой кнопкой мыши для вызова контекстного меню и выбрать команду *Edit Chart,* после чего появляется окно редактирования компонента (рис.14.3).

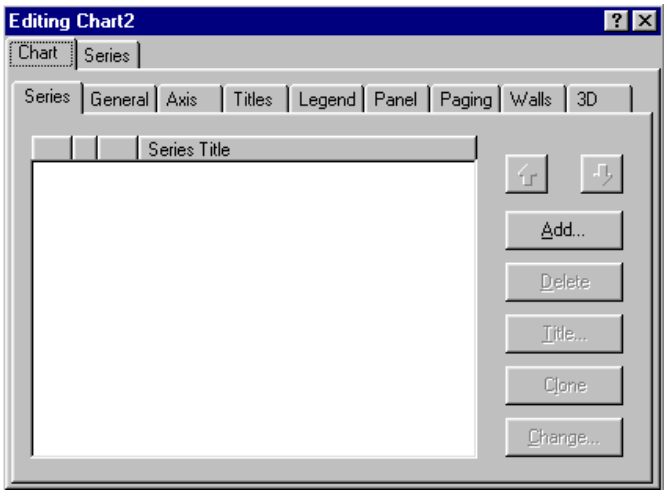


Рис. 14.3. Окно редактирования графиков компонента TChart.

В окне редактирования нужно нажать кнопку *Add* и выбрать подходящий тип графика (рис.14.2). Каждый выбранный график называется серией, которые нумеруются, начиная с нуля. Доступ к сериям осуществляется с по-

мощью закладки *Series*.Окно редактирования имеет закладки для задания заголовка графика Title, редактирования осей – Axis.

После закрытия окна редактора компонент будет содержать примерный вид графика. Однако его реальный вид зависит от фактических данных, которые создаются в работающей программе. В программе обращаются к свойству *SeriesList[<номер серии>]* – список серий; первая созданная серия имеет индекс 0, вторая 1 и т.д. Свойству *SeriesList* доступны методы: *AddX* – добавить данные по X, *AddY* – добавить данные по Y, *AddXY* – добавить данные по X и Y.

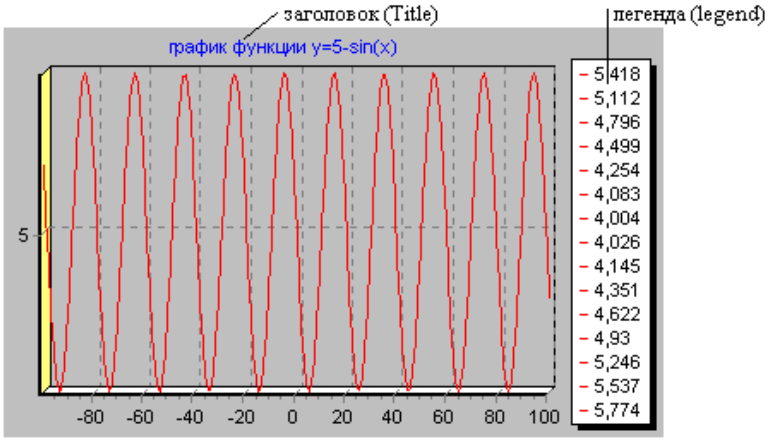


Рис. 14.4 Построение графика функции с помощью компонента TChart.

Например, обработчик события OnCreate формы создает график y=5-sin(x), показанный на рис. 14.4 (график будет показан при открытии окна программы):

```
procedure TForm1.FormCreate(Sender: TObject);
const PI=3.14;
var x:integer; y:real;
begin
 for x:= -100 to 100 do begin
                        y:= 5-sin(x/pi);
                       Chart1.SeriesList[0].AddXY(x,y);
                        end;
end;
```

Тема 15. **Компоненты страница Dialogs**

На странице Dialogs представлены компоненты для вызова стандартных диалогов Windows:

TOpenDialog – выбрать файл

TSaveDialog – сохранить файл

TFontDialog – настроить шрифт

TColorDialog – выбор цвета

TPrintDialog – печать

TPrinterSetupDialog – настройка принтера

TFindDialog – поиск строки

TReplaceDialog – поиск с заменой

Работа со стандартными диалоговыми окнами осуществляется в три этапа:

1. На форму помещается соответствующий компонент и осуществляется настройка его свойств. Сам компонент не виден в момент работы программы (невизуальный компонент), видно лишь создаваемое им стандартное окно.

2. Осуществляется вызов стандартного для диалогов метода Execute, который создает и показывает на экране диалоговое окно. Execute – логическая функция, возвращает в программу True, если результат диалога с пользователем был успешным.

3. Анализ результата метода Execute, использование введенных с помощью диалогового окна данных – имени файла, настроек принтера, выбранного шрифта и т.д.

15.1. *Компонент TColorDialog – выбор цвета.*

Компонент создает и обслуживает стандартное окно выбора цвета, показанного на рис. 15.1.

Рис. 15.1. Окно выбора цвета, создаваемое компонентом TColorDialog.

Свойство компонента *Color:TColor* содержит выбранный цвет в окне диалога.

Пример использования диалога в программе может быть следующий:

```
If ColorDialog1.Execute   {вызов окна диалога}
        then {фигуру залить цветом, выбранным в окне диалога}
            Shape1.Brush.Color := ColorDialog1.Color;
```

15.2. ***Компонент TFontDialog – диалог выбора шрифта.***

Компонент обслуживает стандартное окно выбора шрифта (рис. 15.2).

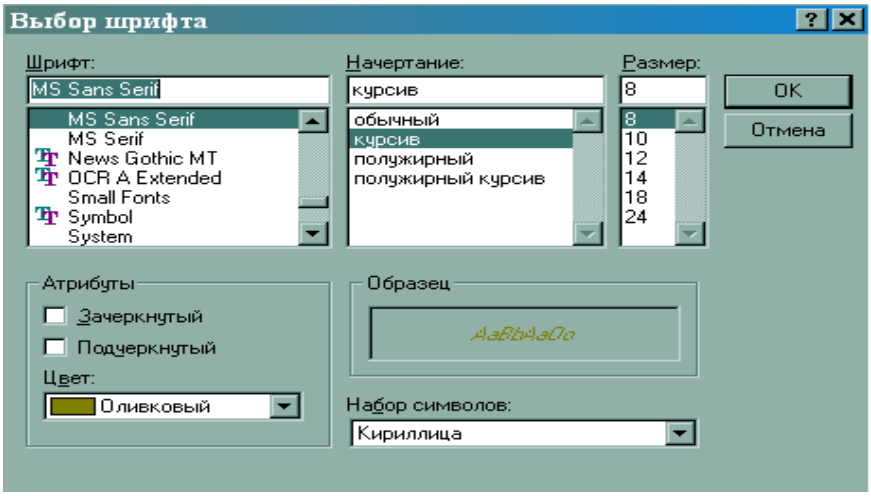


Рис. 15.2. Окно выбора шрифта, создаваемое компонентом TFontDialog.

Свойство компонента *Font: TFont* содержит выбранные параметры шрифта в окне диалога.

Пример использования диалога в программе может быть следующим:

```
If FontDialog1.Execute   {вызов окна диалога}
    Then  {в текстовой области установить выбранные параметры шрифта}
        memo1.Font := FontDialog1.Font;
```

15.3. ***Компоненты TOpenDialog и TSaveDialog – диалоги открытия и сохранения файлов.***

При открытии или сохранении файла пользователь в окне диалога указывает путь и имя файла, возвращаемое свойством *FileName: string*. При открытии файла пользователь может ввести произвольное имя и, следовательно, указать несуществующий файл. Чтобы избежать этого, можно проверить существование файла функцией *FileExists:Boolean*.

Свойство *Filter:string* используется для фильтрации (отбора) файлов, показываемых в диалоговом окне. Это свойство можно устанавливать в программе, например:

```
OpenDialog1.Filter :='текстовые файлы|*.txt|файлы Object Pascal|*.pas';
```

или в окне Инспектора Объектов при выборе свойства Filter, открывающего окно специального редактора (рис.15.3)

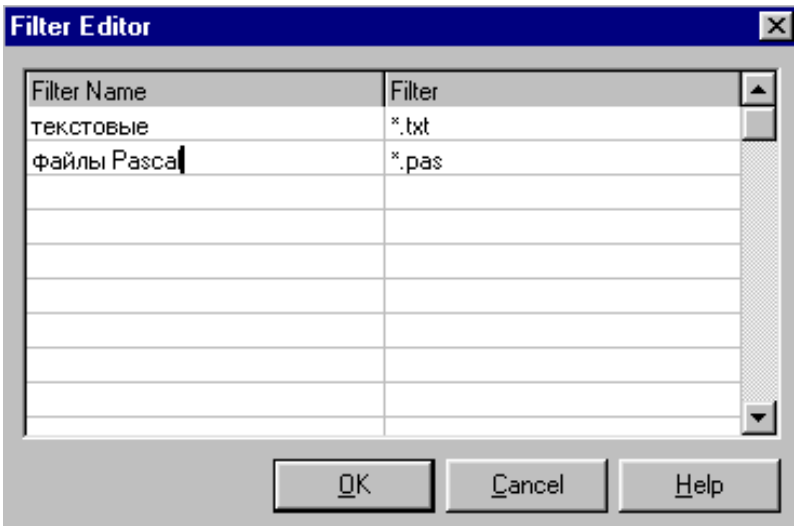


*Рис. 15.3.*Окно редактора фильтра свойства Filter.

Свойство *Title:string* задает заголовок диалогового окна.

Установить начальный каталог окна диалога можно с помощью свойства *InitialDir:string* в программе или в окне Инспектора Объектов.

В свойстве *DefaultExt:string[3]* указывается расширение, присваиваемое сохраняемому файлу по умолчанию (если пользователь не указал другое).

Пример использования компоненты OpenDialog для чтения информации с файла и записи её в memo1:

```
begin
{устанавливаем начальный каталог в окне диалога "Открытие файла"}
Opendialog1.InitialDir:='c:\мои документы';
{Настраиваем диалог на отбор текстовых файлов}
OpenDialog1.Filter:='текстовые файлы|*.txt|файлы Object Pascal|*.pas';
{выполняем диалог и проверяем существование выбранного диалогом файла}
If OpenDialog1.Execute and FileExists(OpenDialog1.Filename)
        then    {считываем содержимое файла в memo1}
       memo1.Lines.LoadfromFile(OpenDialog1.FileName);
end;
```

Пример использования компоненты SaveDialog для записи содержимого компоненты memo1 в файл:

```
begin
SaveDialog1.InitialDir:='c:\мои документы';
SaveDialog1.Filter:='текстовые файлы|*.txt|файлы Object Pascal|*.pas';
SaveDialog1.DefaultExt:='txt'; {указываем расширение файла по умолчанию}
If  SaveDialog1.Execute
        then    {записываем содержимое memo1 в файл}
            memo1.Lines.SaveToFile(SaveDialog1.FileName);
end;
```

В приведенных выше двух примерах рассматривается работа с компонентами, имеющими свойства типа TStrings, и доступные в данном случае

методы SaveToFile, LoadFromFile. При работе с остальными компонентами для записи в файл и чтения информации с файла в Object Pascal существуют специальные функции и методы, описанные в разделе "Файлы" (см. тема 11).

15.4. Компоненты TOpenPictureDialog и TSavePictureDialog – диалоги открытия и сохранения изображений.

Специализированные диалоги для открытия и сохранения графических файлов отличаются от TOpenDialog и TSaveDialog двумя обстоятельствами. Во-первых, в них предусмотрены стандартные фильтры для выбора графических файлов (с расширением bmp,ico,wmf,emf), т.е. отсутствует свойство Filter. Во-вторых, в окна диалога включена панель для предварительного просмотра выбираемого файла.

Считывать содержимое графического файла можно только в компоненты, работающие с графикой, рисунками (Image и др.)

Вызов диалогов происходит программно, например:

```
OpenPictureDialog1.InitialDir:='C:\мои документы';
If OpenPictureDialog1.Execute and FileExists(OpenPictureDialog.FileName)
    then Image1.Picture.LoadFromFile(OpenPictureDialog1.FileName);
```

ЛИТЕРАТУРА

1. Архангельский А.Я. Программирование в Delphi 4. М.: ЗАО «Издательство БИНОМ», 1999 г. 768 с.
2. Дэн Оузьер и др. Delphi 3. Освой самостоятельно/ Пер. с англ. М.: «Издательство БИНОМ»,1998 г. 560 с.
3. Фаронов В.В. Delphi 5. Учебный курс. М.: Нолидж, 2000.608 с.