КАЗАХСКО - АМЕРИКАНСКИЙ УНИВЕРСИТЕТ

С.Б. Дубовиченко

ПРОГРАММИРОВАНИЕ

Pascal и Delphi

Алматы
2003

Казахско - Американский Университет

С.Б. Дубовиченко

ПРОГРАММИРОВАНИЕ

Pascal и Delphi

Учебник по информатике
для ВУЗов

Алматы
2003

УДК 378(075.8): 681.14
ББК 32.973.26-018.1Я7
Д 79

Печатается по решению Научно - методического
Совета Казахско - Американского Университета

***Рецензенты*:**
профессор подготовительного факультета иностранных
граждан Каз.НУ **Аккушкарова К.А.**,
декан факультета компьютерных наук КАУ,
доцент **Байзылдаева У.Б.**

Дубовиченко С.Б.

Д 79 Программирование. Pascal и Delphi: Учебник по информатике для ВУЗов. - Алматы: Изд. Каз. - Амер. Ун., 2003г. - 242с.

ISBN 9965-9308-1-3

В настоящей книге рассматривается программирование на алгоритмическом языке Pascal - 6.0 и работа в системе разработки приложений Delphi - 6.0, которая позволяет создавать пользовательский интерфейс любой прикладной программы, используя язык программирования Object Pascal.

Книга может быть использована студентами ВУЗов, и любыми желающими изучить методы работы на персональном компьютере, основные компьютерные программы и системы.

Д $\frac{060700000}{00(05)-03}$ ББК 32.973.26-018.1Я7

ISBN 9965-9308-1-3

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В настоящей книге дается краткое изложение языка программирования Turbo Pascal - 6.0, который разработан, чтобы удовлетворить требованиям всех пользователей IBM PS/2, PC и совместимых с ними компьютеров. Это алгоритмический язык высокого уровня, который вы можете использовать для написания программ любого типа и размера. Мы приводим описание самого языка программирования и интегрированной среды разработки, которую он использует при своей работе.

Далее приводятся основные методы работы в системе Delphi - 6.0, которая является совершенно новой системой программирования для персональных компьютеров с огромными и совершенно уникальными свойствами и характеристиками. Первая версия системы, выпущенная компанией Borland, явилась результатом разработки, которая велась в обстановке полной секретности в течение нескольких лет. В настоящее время выпущена уже шестая версия этой системы, которую мы и будем рассматривать в данной книге.

Кроме Delphi, у компании Borland появились и другие замечательные продукты (компьютерные системы или программы), так же, как и Delphi, основанные на новых, недавно появившихся у компании технологиях. Это новые программы для BDE 2.0, BC++ 4.5, Paradox for Windows 5.0, dBase for Windows 5.0, BC++ 2.0 for OS/2. Тем не менее, именно Delphi стал именно той системой, на примере которой стало ясно, что один единственный продукт может настолько удачно сочетать несколько передовых технологий.

В данной книге мы опишем основные и наиболее простые возможности новой программной системы компании Borland, а также некоторые особенности проектирования приложений с его помощью. Диапазон разработанных при помощи Delphi программных продуктов очень широк - от игровых программ до мощнейших банковских систем.

Мы приведем описание основных методов управления проектом, создания Форм и некоторые настройки всей системы Delphi. Далее будет приведен обзор стандартных и дополнительных компонент из Палитры Компонент Delphi (Standard и Additional), страницы диалогов (Dialogs) и системных компонент (System).

Вы узнаете о том, какие возможности есть у Delphi для создания приложений, использующих графику, как использовать компоненты для отображения картинок, какие средства есть в Delphi для оформления программ. Далее вы познакомитесь с важным свойством Canvas, которое предоставляет доступ к графическому образу объекта на экра-

не.

Кроме того, мы расскажем о возможных способах вывода информации на печать из программы, созданной в Delphi. Рассмотрим вывод документа в текстовом режиме принтера, вывод графики с помощью объекта TPrinter и печать содержимого Формы.

Автор выражает искреннюю благодарность А.А. Кусаинову, А.Ш. Куанышеву и Г.Б. Халеловой за постоянное содействие при издании данной книги.

TURBO ПАСКАЛЬ

Система программирования Turbo Pascal - 6.0 построена на основе стандартного Паскаля и полностью совместима с более ранними версиями Turbo Pascal. Новая версия включает следующие возможности:

1. Новую интегрированную среду разработки (интерфейс пользователя, редактор для написания программ и компилятор).
2. Множество накладывающихся окон (подобно многооконному режиму Windows).
3. Поддержка мышки, меню, диалоговых окон.
4. Файловый редактор, который может редактировать файлы до 1Мб.
5. Расширенные возможности отладки.
6. Полное сохранение и восстановление среды разработки.
7. Личные поля и методы в объявлении объектов.
8. Директива расширенного синтаксиса, которая позволяет вам интерпретировать функции, как процедуры.
9. Адресные ссылки в константах.
10. Редактирование инициализированных данных из объектных файлов.
11. Более быстрый монитор, сокращающий фрагментацию.
12. Расширенные возможности встроенной справочной системы с возможностью вырезки и вставки кодов (программ) примеров для каждой библиотечной процедуры и функции.

В первой главе мы рассмотрим основные понятия, и приемы программирования на Turbo Pascal, а следующие главы будут посвящены модулям этого языка, управлению всем проектом и работе в интегрированной среде разработки программ.

ПРОГРАММИРОВАНИЕ НА TURBO PASCAL

Язык Паскаль был разработан в начале 70 - х годов, как язык начального обучения программированию. Если у вас есть опыт программирования на других языках, то вам будет не трудно его освоить.

Элементы программирования

Большинство программ, создаются для решения какой - нибудь

конкретной задачи. Решение задачи достигается обработкой информации или некоторых данных. Поэтому, как программисту, вам необходимо знать, как:

1. Ввести информации в программу - ввод данных.
2. Сохранять информацию в программе - данные.
3. Задать правильные команды обработки данных - операции.
4. Получить данные из программы - вывод.

Вы можете написать и упорядочить свои команды так, чтобы:

1. Некоторые из них выполнялись при исполнении некоторого условия или ряда условий - условное выполнение.
2. Выполнялись некоторое число раз - циклы.
3. Собирались в отдельные части, которые могут быть выполнены в нескольких местах программы - подпрограммы.

Итак, перечислены семь основных понятий программирования - ввод, данные, операции, вывод, условное выполнение, циклы и подпрограммы. Этот список неполный, но он содержит основные понятия, присущие всем программам. Многие языки программирования, включая Паскаль, имеют свои особенности. Но когда вы хотите быстро изучить новый язык, то достаточно посмотреть, как он реализует эти семь элементов и начать работать.

Теперь более подробно остановимся на этих понятиях и представлениях.

Ввод

Ввод - это информация (числовые или символьные данные), поступающая с клавиатуры, диска (дискеты) или порта ввода/вывода.

Данные

Данные - это константы, переменные и структуры, содержащие числа (целые и вещественные), текст (символы и строки) или адреса (переменных и структур).

Операции

Операции осуществляют присваивание значений, вычисление выражений (сложение, деление и т.д.), сравнение значений (равно, не

равно, больше и т.д.).

Вывод

Вывод означает вывод и запись информации на экран, на диск или в порт ввода/вывода.

Условное выполнение

Это выполнение одной или набора команд, если выполняется (истинно) некоторое условие (а, если условие не выполняется, то эти команды пропускаются и выполняется другой набор команд) или если элемент данных имеет указанное значение или диапазон значений.

Циклы

В циклах набор команд выполняется определенное число раз, пока истинно некоторое заданное условие или пока условие не станет истинным.

Подпрограммы

Набор инструкций, объединенных именем, которые выполняются в любом месте программы, где есть их вызов по заданному имени.

Типы данных

В процессе программирования, программист имеет дело с информацией, представляющей пять основных типов данных: целые числа, вещественные числа, символы и строки символов, булевские данные и указатели:

1. Целые числа - это числа, с помощью которых вы учились считать в школе (1, 5, - 21, 752 и т.д.).
2. Вещественные числа имеют дробные части (3.14159) и экспоненты (2.579x10+24). Они известны так же, как числа с плавающей точкой.
3. Символы - это любые буквы алфавита, символы и цифры от 0 до 9. Они могут использоваться отдельно (a, z, !, 3) или объединяться в символьные строки ('Это только проверка').
4. Булевские выражения имеют только два значения: TRUE или

FALSE (ИСТИНА или ЛОЖЬ) и используются в условных выражениях.

5. Указатели - это адреса ячеек памяти, содержащих некоторую информацию.

Целые числа

В стандартном Паскале целочисленный тип определяется в пределах от -MaxInt до +MaxInt, где MaxInt - набольшее возможное целое значение, допустимое для процессора (и компьютера в целом).

В Turbo Pascal поддерживается целочисленный тип с MaxInt = 32767, допуская и значение -32768. Переменная целочисленного типа занимает 2 байта. Кроме того, поддерживается четыре других целочисленных типов данных, каждый из которых имеет свой диапазон значений.

Тип	***Диапазон***	***Размер в байтах***
Byte	0 . . 255	1
Shortint	- 128 . . 127	1
Integer	- 32768 . . 32767	2
Word	0 . . 65535	2
Longint	- 2147483648 . . 2147483647	4

Turbo Pascal позволяет использовать шестнадцатеричные целые значения (основание числа равно 16). При описании шестнадцатеричной константы перед ней указывается знак доллара $, например $A=39.

Вещественный тип данных

В стандартном Паскале тип Real представляет значение с плавающей точкой, содержащее мантиссу и экспоненту, а степень числа равна 10. Количество значащих цифр в мантиссе и диапазон значений экспоненты зависят от типа компьютера.

В Turbo Pascal данные вещественного типа Real имеют размер 6 байт с 11 значащими цифрами мантиссы и интервалом экспоненты от 10^{-39} до 10^{38}. Кроме того, Turbo Pascal поддерживает стандарт для двоичной арифметики с плавающей точкой. В этом случае добавляются типы данных Single, Double, Extended и Comp:

1. Single - размер 4 байта, допускается 7 значащих цифр и диапа-

зон экспоненты от 10^{-45} до 10^{38};

2. Double - размер 8 байт, допускается 15 значащих цифр и диапазон экспоненты от 10^{-324} до 10^{308};

3. Extended - размер 10 байт, допускается 19 значащих цифр и диапазон экспоненты от 10^{-4951} до 10^{4931}.

Если у вашего компьютера 86 процессор или выше, например, 386 Pentium и включена работа с числовым сопроцессором, Turbo Pascal генерирует инструкции для поддержки этих типов и выполнения всех операций с плавающей точкой.

Тип	***Диапазон***	***Размер в байтах***
Real	2.9E-39 . . 1.7E38	6 (вещественный)
Single	- 1.5E-45 . . 3.4E38	4 (с одинарной точностью)
Double	5.0E-324 . . 1.7E308	8 (с двойной точностью)
Extended	- 1.9E-4951 . . 1.1E4932	10 (повышенной точности)
Comp	- 2E63 +1 . . 2E63	8 (сложный - только целые значения)

Вызовем редактор Turbo Pascal и наберем следующую программу (каждая строчка с операторами заканчивается знаком "точка с запятой" - ";" , а последний оператор программы end заканчивается точкой - "."):

```
program Ratio;
var
A, B: Integer;
R: Real;
begin
Write('Enter two numbers: ');
Readln(A, B);
R: = A div B;
Writeln('The ratio is ', R);
end.
```

С помощью функции основного меню интерфейса File/Save As сохраним ее в файле RATIO.PAS. Нажмите далее ALT + R для компиляции и запуска программы. Программа потребует задания двух чисел (на экране появится вопрос) - введем два значения - 10 и 3, и на экране получим (увидим) результат работы программы 3.000000.

Хотя мы ожидали ответа 3.3333333333, который получается при делении 10 на 3, мы получили результат 3. Это произошло потому, что был использован оператор div для деления целых чисел, который просто отбрасывает дробную часть, получившуюся при делении.

Заменим оператор div на оператор обычного деления:

```
R: = A/B;
```

сохраним новый код программы (клавишей F2), откомпилируем ее и снова выполним. В результате получим 3.3333333333, как и следовало ожидать.

Символьные и строковые типы данных

Научившись записывать числовые данные, научимся записывать и использовать символьные и строковые данные. Паскаль позволяет определять тип Char, имеющий размер в один байт и содержащий один символ. Символьные константы содержат один символ, заключенный в апострофы ('A','e','r','2'). Заметим, что '2' означает символ 2, а 2 означает целое число два (и 2.0 - вещественное число).

Рассмотрим пример программы:

```
program Ratio;
var
A, B: Integer;
R: Real;
Ans: Char;
begin
repeat
Write ('Enter two numbers: ');
Readln (A, B);
R: = A / B;
Writeln ('The ratio is ', R);
Write ('Do it again? (Y/N)');
Readln (Ans);
until UpCase(Ans) = 'N';
end.
```

После вычисления коэффициента, выдается сообщение:

Do it again? (Y/N) - Повторить? (Да/Нет)

Программа находится в состоянии ожидания одного из указанных символов и нажатия клавиши "Enter". Если вы нажмете заглавную "N", то условие "until" будет выполнено и цикл закончится. В случае если вы ответите "Y" или "y", то выполнение цикла будет продолжено. Заглавная "N" и маленькая "n" не одно и тоже, потому что они имеют различные значения в коде ASCII. Каждый символ имеет свой код ASCII, представленный 8 - битным значением (один символ занимает 1 байт или 8 бит).

В Turbo Pascal существуют два дополнительных способа описания символьных констант:

1. С помощью символа "^".
2. Символа числа "#".

Символы с кодами от 0 до 31 являются управляющими символами. Они обозначаются аббревиатурами (CR - возврат каретки, LF - перевод строки, ESC - выход) или с помощью комбинации двух клавиш, например, Ctrl + буква (управляющий символ кода ASCII, обозначаемый числом 7 известен, как Bel).

Turbo Pascal дает возможность представить символы с помощью значка "^", за которым следует буква (или символ). Так, "^G" то же самое, что и Ctrl + G. Обозначение "^G" можно использовать в операторах Turbo Pascal:

Writeln (^G) - Этот метод применим только к управляющим символам.

Вторая возможность обозначения - использование символа номера "#", за которым следует код ASCII. Так, #7 то же самое, что и "^G", #65 обозначает символьную величину 'A', а #233 - один из специальных псевдографических символов IBM PC.

В большинстве случаев, в различных программах используются именно строки символов. В стандартном Паскале тип строк символов не поддерживается, а в Turbo Pascal эта возможность реализована оператором String(30). Рассмотрим программу:

```
program Hello;
var
Name: String(30);
```

```
begin
Write ('What is your name? ');
Readln (Name);
Writeln ('Hello, ', Name)
end.
```

Переменная Name объявлена, как String (строка) и под нее резервируется 30 байт (для 30 символов). Кроме того, Turbo Pascal отводит еще один байт, в котором содержится текущая (введенная в данный момент) длина строки. В этом случае, независимо от того, какой длины будет введено имя, оператор Writeln распечатает имя указанной длины. Если будет введено имя больше 30 символов, то будут использоваться только первые 30 символов, а остальные будут проигнорированы.

При описании строковой переменной можно указывать ее размер, но не более 255. "По умолчанию" длина строковой переменной именно 255 символов.

Булевские данные

Встроенный тип Boolean данных в Turbo Pascal имеет два возможных значения: True и False (истина и ложь). Можно объявить переменную типа Boolean и присвоить ей значение True или False или же, что более важно, присвоить ей значение выражения, которое при вычислении принимает одно из этих значений. Булевское выражение это выражение, которое принимает значение True или False и состоит из выражений отношений, булевских операторов, булевских переменных и/или других булевских выражений. Пример:

```
while (Index <= Limit) and not Done do...
```

оператор while содержит булевское выражение. Булевское выражение в этом случае это все, что находится между ключевыми словами while и do. Done это переменная (или функция) булевского типа.

Идентификаторы

До сих пор в примерах содержались идентификаторы переменных без описания ограничений и правил их образования. Теперь рассмотрим их более подробно. Имена, которые даются константам, типам данных, переменным и функциям известны, как идентификаторы. Идентификаторы использовались для образования следующих вели-

чин:

1. Integer, Real, String - встроенные типы данных.
2. Hello, DoSum, Ratio - название основных программ.
3. Name, A, B, Sum, Ratio - переменные.
4. Write, Writeln, Readln - имена встроенных процедур.

Turbo Pascal имеет несколько правил образования идентификаторов:

- Все идентификаторы начинаются с буквы или знака подчеркивания (a . . z, A . . Z, _). Последующими символами могут быть буква, знак подчеркивания, цифра (0 - 9). Другие символы недопустимы.
- Идентификаторы рассматриваются без учета регистров (прописные и строчные буквы не различаются). Это значит, что a . . z тождественно A . . Z. Например, index, то же самое, что Index и INDEX.
- Идентификаторы могут иметь различную длину, но используются только первые 63 символа.

Операторы

Итак, некоторые данные получены вашей программой и присвоены определенным вами переменным. Программе нужно их обработать и получить результат. Именно для этого и используются операторы. Существует восемь типов операторов:

1. Присваивания.
2. Арифметические.
3. Побитовые.
4. Отношений.
5. Логические.
6. Операции над множествами.
7. Адресные.
8. Операции над строками.

Большинство операторов в Паскале бинарные, т.е. используются между двумя операндами (переменными или константами, над которыми выполняются операции). Но имеются и унарные операторы, которые имеют один операнд (располагаются перед одним операндом). Бинарные операторы имеют обычную алгебраическую форму, например, a + b. Унарный оператор предшествует своему операнду, напри-

мер, -b. В сложных выражениях порядок выполнения операций определяется правилами приоритета, которые приведены в следующей таблице.

Операторы	***Приоритет***	***Категория***
@, not	Первый (высший)	Унарные
*, /, div, mod, and, shl, shr	Второй	Мультипликативные
+, - , or, xor	Третий	Аддитивные
=, <>, <, >, <=, >, =, in	Четвертый (низший)	Отношения

Операции равного приоритета выполняются слева направо, хотя компилятор может их перегруппировать для генерации оптимального кода. Выражения, заключенные в скобки, вычисляются в первую очередь, независимо от предшествующих и последующих операторов.

Операторы присваивания

Основной операцией в любых программах является операция присваивания:

R: = A/B

В Паскале операция присваивания это комбинация двоеточия и знака равенства ":=". В приведенном примере значение выражения A/B, стоящее справа от знака равенства ":=" присваивается переменной R, стоящей слева.

Арифметические операторы

Паскаль поддерживает обычный стандартный набор бинарных арифметических операторов, которые выполняются над целыми и вещественными числами:

1. Умножение (*).
2. Деление целых (div).
3. Деление вещественных с остатком (/).
4. Остаток от деления (mod).
5. Сложение (+).
6. Вычитание (-).

Кроме того, поддерживаются унарные операторы следующего типа:

1. Унарный минус (a + (-b)).
2. Унарный плюс (a + (+b)).

Побитовые операторы

Для операций над битами в Паскале имеются следующие операторы:

1. shl (shift left) - сдвигает биты влево на указанное число бит, заполняя оставшиеся справа разряды нулями.
2. shr (shift right) - сдвигает биты вправо на указанное число бит, заполняя оставшиеся слева разряды нулями.
3. and - выполняет логическое and (и) над парой битов, возвращая 1, если оба бита 1 и 0 в противном случае.
4. or - выполняет логическое or (или) над парой битов, возвращая 0, если оба бита равны 0 и 1 в противном случае.
5. xor - выполняет логическое исключающее "или" над парой битов, возвращая 1, если биты имеют разное значение и 0 в противном случае.
6. not - операция логического дополнения бита заменяет 0 на 1 и обратно.

Эти операции выполняют действия на низком уровне с целочисленными значениями.

Операторы отношений

Операторы отношений сравнивают два значения, возвращая в результате булевское значение True или False. В Паскале реализуются следующие операторы:

1. > - больше, чем.
2. >= - больше, чем или равно.
3. < - меньше, чем.
4. <= - меньше, чем или равно.
5. = - равно.
6. <> - не равно.
7. in - является элементом (входит в ...).

В некоторых случаях (в различных программах) необходимо знать, каков результат операций отношения - True или False. Введем следующую программу:

```
program TestGreater;
var
A, B: Integer;
T: Boolean;
begin
Write ('Введите два числа: ');
Readln (A, B);
T: = A>B;
Writeln ('A больше чем B', T);
end.
```

Результат программы (который выводится на экран) True, если A больше чем B и False, если A меньше или равно B.

Логические операторы

В стандартном Паскале (и в Turbo Pascal) существуют четыре логических оператора:

and, xor, or, not.

Они аналогичны побитовым операторам, но имеют свои отличия. Эти логические операторы работают с логическими значениями (True и False), позволяя комбинировать выражения отношений, булевские переменные и булевские выражения.

Существуют следующие различия между логическими и побитовыми операторами:

1. Логические операторы возвращают результат True или False (булевское значение Истина или Ложь), в то время как побитовые операторы производят действие над целыми значениями. Эти операторы не позволяют комбинировать булевские и целые выражения - другими словами, выражение "Flag and Indx" недопустимо, если Flag - булевский тип, а Indx - целый тип (или наоборот).

2. Логические операторы and и or "по умолчанию" имеют короткую форму вычисления, а xor и not такой формы не имеют. Допустим, имеется выражение "exp1 and exp2". Если exp1 имеет значение False, то

и все выражение имеет значение False, а выражение exp2 не вычисляется. Аналогично, в выражении "exp1 or exp2", exp2 не будет вычисляться, если exp1 имеет значение True.

Адресные операторы

В Паскале поддерживаются два специальных оператора над адресами:

1. Вычисление адреса (@).
2. Оператор косвенной ссылки (^).

Оператор @ возвращает адрес некоторой заданной переменной. Так, если Sum переменная целого типа, то @Sum является адресом этой переменной в памяти компьютера. Аналогично, если ChrPtr это указатель на тип Char, то ChrPtr^ это символ, на который указывает ChrPtr^.

Операторы над множествами

Операторы над множествами выполняются в соответствии с правилами логики теории множеств. Они включают:

1. + - Объединение.
2. - - Разность.
3. * - Пересечение.

Строковые операторы

Существует единственный оператор "+", который выполняет конкатенацию (сложение) двух строк или символьных переменных.

Вывод данных

Возможно, вам покажется странным, что речь о выводе информации пойдет прежде, чем о вводе данных, но программа, которая не выводит какую - либо информацию просто не имеет смысла. Вывод обычно принимает форму, которая зависит от выходного устройства:

1. На экран монитора вашего компьютера (слова и изображения).
2. На запоминающие устройства (дискеты и винчестер - жесткий

диск компьютера).

3. В порты ввода/вывода компьютера.

Процедуры Writeln и Write

В приведенных выше примерах программ мы уже использовали наиболее распространенную функцию Паскаля - подпрограмму Writeln. Назначение ее - запись (вывод) информации на экран. Ее формат прост и гибок при использовании:

```
Writeln (элемент 1, элемент 2, ....);
```

Каждый элемент это то, что вы хотите вывести на экран, а это может быть целое (3, 5, 27) или вещественное число (5,83, -2631.4), символ ('b', 'Y'), строка ('Привет Паскаль') или булевское значение (True или False). Кроме того, это может быть именованная константа (имя константы), переменная, указатель и вызов функции, если она возвращает значение целого типа.

Все элементы печатаются в строку в заданном вами порядке. После вывода всех элементов курсор устанавливается на начало следующей строки. Если есть необходимость оставить курсор в этой же строке после последнего элемента, то используйте оператор:

```
Write (элемент 1, элемент 2, ...);
```

При выводе элементов, сам оператор Writeln пробелы между ними не вставляет. При желании иметь такие пробелы, необходимо их учесть самому:

```
Writeln (элемент 1,' ',элемент 2,' ',....);
```

Приведем примеры вывода, который выполняется оператором Writeln:

```
A:=1; B:=2; C:=3;
Name := 'Frank';

Writeln (A, B, C);
123

Writeln (A, ' ', B, ' ', C);
```

```
1 2 3

Writeln ('Hi', Name);
HiFrank;

Writeln ('Hi, ',', Name, '.');
Hi, Frank.
```

Кроме того, можно использовать параметры определения ширины поля для данного элемента. В этом случае оператор имеет формат:

```
Writeln (элемент: длина, ....);
```

где длина - целое выражение (литерал, константа, переменная, вызов функции), определяющее общий размер поля для вывода элемента. Рассмотрим пример программы и полученный в результате ее работы вывод данных:

```
A:=10; B:=2; C:=100;

Writeln (A, B, C);
102100

Writeln (A:2, B:2, C:2);
10 2100

Writeln (A:3, B:3, C:3);
 10  2100

Writeln (A, B:2, C:4);
10 2 100
```

При использовании длины поля вывода, каждый элемент дополняется слева начальными пробелами (если поле больше самого числа) в соответствии с ее величиной. Само выводимое значение выравнивается вправо.

Во втором операторе Writeln приведенного примера для C=100, длина поля меньше, чем нужно, т.е. задано 2, а нужно 3. В таком случае Паскаль увеличивает размер поля до минимально необходимого.

Этот метод применим для всех допустимых элементов: целого типа, вещественных чисел, символов, строк и булевских типов. Однако,

при указании ширины (размера) поля для вещественных чисел выравнивание происходит слева, а распечатывается они в экспоненциальной форме:

```
x:=421.53;
Writeln (x);
4.2153000000E+02

Writeln (x:8);
4.2E+02
```

Для вещественных чисел Паскаль позволяет добавить второй операнд длины:

```
элемент : длина : количество цифр
```

Вторая длина (количество цифр) указывает, сколько цифр нужно выводить для числа с фиксированной точкой после самой точки:

```
x:=421.53;
Writeln(x:6:2);
421.53

Writeln(x:8:2);
  421.53

Writeln(x:8:4);
421.5300
```

Ввод данных

В стандартном Паскале есть две основных функции ввода информации Read и Readln, которые используются для чтения данных с клавиатуры.

Их формат имеет вид:

1. Read (элемент 1, элемент 2, ...);
2. Readln (элемент 1, элемент 2, ...);

где каждый элемент это переменная целого, вещественного, символьного типа или строка. Числа при вводе должны отделяться друг от

друга пробелами или нажатием клавиши Enter.

Условные операторы

Иногда бывает необходимо выполнить часть программы, если заданное условие имеет значение True или False, или когда заданное выражение принимает определенное значение. Рассмотрим, как это реализуется в Паскале, в том числе и в Turbo Pascal версии 6.0.

Оператор IF

Оператор if имеет следующий общий формат:

```
if выражение
then оператор 1
else оператор 2,
```

где выражение - любое булевское выражение (вырабатывающее в результате True или False), "оператор 1" и "оператор 2" - любые операторы Паскаля. Если выражение принимает значение True, то выполняется "оператор 1", в противном случае - "оператор 2".

Имеются два важных момента, на которые следует обратить внимание при использовании оператора if then else. Во - первых, оператор else обязательным не является, т.е. допустимо использовать оператор if в следующем виде:

```
if выражение
then оператор 1
```

В этом случае "оператор 1" выполняется только тогда, когда выражение имеет значение True. В противном случае "оператор 1" пропускается и выполняется следующий за ним оператор.

Во - вторых, если необходимо выполнить более одного оператора, когда выражение принимает значение True или False, то следует использовать составной оператор. Составной оператор это ключевое слово begin, несколько операторов разделенных точкой с запятой и ключевое слово end.

Приведем теперь пример использования одного исполняемого оператора:

```
if B = 0.0 then
```

```
Writeln ('Деление на нуль невозможно.')
```

А теперь пример использования составного оператора в предложении else (в части then оператора if все делается точно также):

```
else
begin
R: = A / B;
Writeln ('Отношение = ', R)
end;
```

Циклы

В случае, когда при выполнении какого - либо условия (или невыполнения) необходимо выполнять группу операторов повторно, используются циклы.

Существует три основных вида циклов:

1. while.
2. repeat.
3. for.

Цикл WHILE

Цикл while используется для проверки некоторого условия в самом начале цикла. Общий формат оператора цикла while имеет вид:

```
while выражение do
оператор;
```

В цикле while вычисляется заданное "выражение". Если оно имеет результатом True, выполняется "оператор" (который может быть и группой операторов). В противном случае выполнение цикла завершается и выполняется следующий за ним оператор программы.

В качестве примера приведем следующую программу:

```
program Hello;
var
C: Integer;
begin
C: = 1;
```

```
While (C <= 10) do
begin
Writeln ('Проверка цикла');
C: = C + 1;
end;
Writeln ('Конец цикла');
end.
```

С начала переменной "С" присвоится значение равное 1. Затем, при входе в цикл проверяется "условие" - значение "С" меньше или равно 10. Если это условие выполняется, то выполняется тело цикла (операторы, находящиеся между ключевыми словами begin . . . end). На экран выводится сообщение "Проверка цикла". Затем значение "С" увеличивается на 1 и происходит возврат в начало цикла. Заново проверяется значение "С" и тело цикла выполняется вновь, до тех пор, пока значение переменной "С" удовлетворяет условию. Как только значение "С" становится равным 11, цикл завершается и на экран выводится сообщение "Конец цикла".

Цикл REPEAT .... UNTIL

Общий формат этого оператора цикла:

```
repeat
оператор;
оператор;
. . .
оператор;
until выражение;
```

Существуют несколько основных отличий от цикла while:

1. Операторы в цикле repeat выполняются хотя бы один раз, потому что проверка "выражения" осуществляется в конце тела цикла. В цикле while, если значение "выражения" равно False, тело цикла пропускается сразу.

2. Цикл repeat выполняется до тех пор, пока "выражение" не станет равным True, в то время как цикл while выполняется до тех пор, пока "выражение" имеет значение True. При замене одного типа цикла на другой необходимо на это обращать особое внимание.

Цикл repeat ... until рассмотрим на примере программы RATIO.PAS:

```
program Ratio;
var
A, B: Integer;
R: Real;
An: Char;
begin
repeat
Write ('Введите два числа');
Readln (A, B);
R: = A/B;
Writeln ('Отношение равно', R);
Writeln ('Повторить? (Y/N)');
Readln (An);
until UpCase(An) = 'N';
end.
```

Как говорилось ранее, выполнение операторов в этой программе повторяется, пока ответ на вопрос - "N" (Повторить? Y/N). Другими словами repeat и until, повторяются, до тех пор, пока значение "выражения" при until не будет равно True.

В качестве примера рассмотрим теперь программу Hello, где цикл while заменен на цикл repeat:

```
program Hello;
var
C: Integer;
begin
C := 1;
repeat
Writeln ('Проверка цикла');
C: = C + 1;
until C > 10;
Writeln ('Конец цикла');
end.
```

Отметим, что теперь переменная "C" проверяется на значение больше 10, а в цикле while было условие "C" <= 10. В цикле repeat может использоваться просто группа операторов, а не составной опера-

тор, заключенный между begin ... end, как это было для цикла while.

Цикл FOR

Цикл for существует во многих языках программирования, в том числе и в Паскале. Обычно, набор операторов выполняется фиксированное число раз, пока переменная (индексная) принимает значение в указанном диапазоне.

Модифицируем, приведенную выше программу Hello следующим образом:

```
program Hello;
var
C: Integer;
begin
for C: = 1 to 10 do
Writeln ('Проверка цикла');
Writeln ('Конец цикла');
end.
```

Из этой программы видно, что цикл for выполняется так же, как циклы while и repeat, и фактически эквивалентен циклу while. Общий формат цикла for имеет вид:

```
for индекс: = выражение 1 to выражение 2 do
оператор;
```

где индекс - заданная переменная (целого типа, символьного, булевского типа), "выражение 1" и "выражение 2" - выражения типа, совместимого с типом индекса, оператор - одиночный или составной оператор.

После каждого выполнения цикла его индекс увеличивается на 1. Индекс можно уменьшать на 1 - для этого ключевое слово to заменяется на downto. Цикл for эквивалентен следующей программе с циклом while:

```
i: = exprl;
while i <= expr2 do
begin
оператор;
i: = i + 1;
```

```
end;
```

Главный недостаток цикла for это возможность уменьшить или увеличить индекс только на 1. Основные преимущества - краткость, возможность использования символьного и перечислимого типов в диапазоне значений.

Процедуры и функции

Рассмотрим теперь, как можно выполнить один и тот же набор команд в разных местах программы и с разными данными. Эту группу операторов можно объединить в подпрограмму, которую можно вызывать по необходимости в разных местах основной программы.

В Паскале есть два вида подпрограмм - процедуры и функции. Главное различие между ними заключается в том, что функция возвращает значение через свое имя и может быть использована в выражении типа:

```
X: = sin(A);
```

в то время как процедура может быть вызвана только таким образом:

```
writeln ("Проверка");
```

Однако перед подробным знакомством с процедурами и функциями, необходимо вначале рассмотреть общую структуру программ на Паскале.

Структура программ

В стандартном Паскале программы имеют следующий жесткий формат:

```
program имя программы;

label
метки;

const
объявление констант;
```

```
type
определение типов данных;

var
объявление переменных;
procedure или function;

begin
тело программы;
end.
```

Наличие всех пяти секций объявлений - label, const, type, var, procedure или function в каждой программе необязательно. Однако для стандартного Паскаля, если они присутствуют, порядок их следования строго регламентирован и в программе они должны присутствовать только один раз. За секцией объявлений, следуют процедуры и функции, и только затем тело программы.

Turbo Pascal обеспечивает более гибкую структуру своих программ. Главное - это оператор program должен быть первым, а тело программы последним. Порядок описания остальных секций жестко не регламентирован, но идентификаторы должны быть объявлены до их использования во избежание ошибок компиляции.

Структура процедуры и функции

Процедуры и функции, известные под общим именем, как подпрограммы могут быть описаны в любом месте программы, но до тела главной программы. Общий формат процедур имеет вид:

```
procedure имя процедуры (параметры);

label
метки;

const
объявление констант;

type
определения типов данных;
var
```

```
объявления переменных;
procedure и function;

begin
тело главной процедуры;
end;
```

Функции имеют такой же формат, как и процедуры, только они начинаются с заголовка function и заканчиваются типом данных возвращаемого значения:

```
function имя функции (параметры): тип данных;
```

Имеются только два различия между процедурами и функциями:

1. Процедуры и функции имеют заголовок procedure или function соответственно, а не program.
2. Процедуры и функции заканчиваются точкой с запятой (;), а не точкой (.).

Процедуры и функции могут иметь описания своих констант, типов данных, переменных и свои процедуры и функции. Но все эти элементы могут быть использованы только в тех процедурах и функциях, где они объявлены.

Пример программы

Рассмотрим версию программы Ratio, в которой используется процедура получения двух значений и функция, определяющая их отношение:

```
program Ratio;
var
A, B: Integer;
R: Real;

procedure Get(var X, Y: Integer);
begin
Writeln ('Введите два числа:');
Readln (X, Y);
end;
```

```
function GetR(I, J: Real): Real;
begin
GetR=I/J;
end;

begin
Get(A, B);
R: = GetR(A, B);
Writeln ('Отношение равно ', R);
end.
```

Это, конечно, не улучшение первоначальной программы (приведенной выше), так как она имеет больший размер и медленнее выполняется. Но она показывает, как используются и работают процедуры и функции.

После компиляции и запуска программы первым выполняется оператор Get(A,B). Этот тип оператора известен, как вызов процедуры. При обработке вызова выполняются операторы внутри Get, при этом X и Y (формальные параметры) заменяются на A и B (фактические параметры). Ключевое слово var перед X и Y в операторе процедуры Get говорит о том, что фактические параметры должны быть переменными и что значения переменных могут быть изменены и возвращены вызывающей программе. При завершении работы Get управление возвращается в главную программу на оператор, следующий за вызовом Get.

Следующий оператор - вызов функции GetR. Отметим некоторые отличия - во - первых, GetR возвращает значение, которое должно быть использовано. В данном случае, оно присваивается переменной "R". Во - вторых, значение GetR присваивается в главной программе и этим функция определяет, какое значение возвращается.

В - третьих, нет ключевого слова var перед формальными параметрами I и J. Это означает, что они могут быть любыми выражениями, например, такими как R: = GetR(A+B, 300). Причем, если их значения будут изменены внутри функции, то новые значения не возвратятся обратно в вызывающую программу. Однако это не является отличием процедуры от функции. Можно использовать оба типа параметров для обоих типов подпрограмм.

Комментарии

Иногда бывает необходимо вставить в программу замечания, напоминающие или информирующие о том, что означает переменная,

какие действия выполняет функция или оператор. Эти замечания называют комментариями, и Паскаль позволяет вставлять в программу сколько угодно комментариев.

Комментарий начинается левой фигурной скобкой ({), которая указывает компилятору игнорировать все, что стоит за ней, до тех пор, пока не встретится правая фигурная скобка (}). В программе комментарий может занимать и несколько строк подряд:

```
{Это пример длинного комментария,
занимающего несколько строк}
```

Кроме того, существует альтернативная форма комментария, начинающаяся с символа "(*" и заканчивается символом "*)". Комментарий, начинающийся с "(*", игнорирует все фигурные скобки и наоборот.

МОДУЛИ TURBO PASCAL

Читая предыдущую главу, вы научились писать простые программы на Паскале (а именно на Turbo Pascal), познакомились и переменными, процедурами и функциями. Но как быть в случае нестандартного программирования на персональном компьютере, связанного с управлением работой экрана, вызовами операционной системы MS DOS и различной графикой (рисунками, графиками).

Для того чтобы писать такие программы необходимо иметь понятия о модулях языка Turbo Pascal (просто подпрограмм уже не достаточно). В этой главе описано, что такое модуль, как его использовать, какие модули доступны, как их писать, а затем компилировать.

Модули

В Turbo Pascal возможен доступ к большому числу встроенных констант, типов данных, переменных, процедур и функций. Некоторые из них специфичны именно для Turbo Pascal, другие для IBM PC и совместимых с ним PC (персональный компьютер) в целом или для всей ОС MS DOS (ОС - операционная система). Количество различных программ подобного рода очень велико, но почти никогда все они сразу в программах не используются. Все эти программы разделены на связанные между собой группы, называемые модулями и можно использовать только те модули, которые вам необходимы в данное время.

Модуль это набор констант, типов данных, переменных, процедур и функций. Каждый модуль аналогичен отдельной программе и состоит из главного тела (основной части), которое вызывается перед стартом основной программы и производит необходимые действия по инициализации (присвоении констант, объявлении типов и т.д.), когда это необходимо. Таким образом, каждый модуль это библиотека объявлений, которую можно вставить и использовать внутри основной программы, что позволяет разделить программу на части и компилировать их отдельно.

Объявления внутри модуля связаны друг с другом, например, модуль Crt (смотрите далее) содержит все объявления для программ работы с экраном PC. Turbo Pascal предоставляет восемь стандартных модулей, а шесть из них System, Overlay, Graph, DOS, Crt, и Printer - осуществляют поддержку ваших программ на Turbo Pascal. Все они сохранены в файле TURBO.TPL, который входит в состав самого Turbo Pascal. Два другие модуля - Turbo3 и Craph3 - осуществляют поддержку совместимости программ, написанных для версии Turbo Pascal 3.0.

Структура модуля

Модуль обеспечивает набор некоторых средств для вашей программы, благодаря возможности использования процедур и функций, которые в свою очередь поддерживают константы, типы данных и переменные. Но действительная реализация этих подпрограмм (процедуры и функции) скрыта из - за того, что модуль разделен на два раздела: интерфейс и реализация (именно здесь находятся все подпрограммы). А все объявления и описания модуля (интерфейс) становятся доступными программе, которая его использует.

Структура модуля похожа на структуру программы, но имеет и некоторые отличия:

```
unit <идентификатор модуля>;

interface <интерфейс>
uses <список модулей>; {общие объявления доступные основной программе}

implementation <реализация>
uses <список модулей>; {личные внутренние объявления, которые внешней программе не доступны}
{реализация процедур и функций}

begin
{код инициализации}
end.
```

После заголовка модуля unit следует имя модуля - его идентификатор. Следующий элемент это ключевое слово interface, которое обозначает начало раздела интерфейса модуля, доступного для всех других модулей и программ, использующих этот модуль. В предложении uses указываются модули, которые может использовать этот модуль. Слово uses может появляться в двух местах:

1. Сразу же после слова interface - в этом случае константы или типы данных, объявленные в интерфейсе модуля могут быть использованы в любых других объявлениях и программах.
2. Сразу же после слова implementation - в этом случае любые объявления этого модуля могут использоваться только внутри данного раздела реализации.

Раздел интерфейса

Раздел интерфейса - "открытая" часть модуля (доступная другим программам), которая начинается ключевым словом interface (интерфейс) следующим сразу за заголовком модуля и ограничена ключевым словом implementation (реализация). Интерфейс определяет, что является видимым (доступным) для некоторой другой программы (или других модулей), использующих этот модуль.

Любая программа, использующая этот модуль, имеет доступ к этим видимым, общим элементам. В интерфейсе модуля можно объявить константы, типы данных, переменные, процедуры и функции. Как и в программе, они могут быть расположены в любом порядке, т.е. разделы могут встречаться повторно:

(type...var...<proc>...type...const...var)

Процедуры и функции, доступные для программы, использующей этот модуль, описываются в разделе интерфейса. А их действительные тела (части) - операторы, реализующие эти подпрограммы, размещены в разделе реализации.

Секция реализации

Раздел реализации - "закрытая", недоступная часть модуля, которая начинается со слова implementation. Все, что объявлено в части интерфейса видимо из раздела реализации - это константы, типы, переменные, процедуры и функции. Кроме того, в разделе реализации могут быть свои собственные дополнительные объявления, недоступные программам, использующим этот модуль. Такие программы не могут обращаться и ссылаться на них.

Однако эти недоступные элементы могут использоваться (и, как правило, это делается) видимыми процедурами и функциями, заголовки которых появляются в разделе интерфейса. Предложение uses может появляться и в разделе implementation - в этом случае uses следует непосредственно за ключевым словом implementation.

Если внутри модуля некоторые процедуры были объявлены, как внешние, то в любом месте исходного файла должна быть директива {$L имя файла}. Обычные процедуры и функции, объявленные в разделе интерфейса и не являющиеся встроенными должны появляться в разделе реализации.

Заголовок procedure (или function) в разделе реализации должен

быть такой же, как в разделе интерфейса или записываться в короткой форме. В краткой форме за ключевым словом (procedure или function) следует ее идентификатор (имя процедуры) без перечисления параметров (смотрите пример ниже). Подпрограмма (procedure или function) содержит свои собственные локальные объявления (метки, константы, типы, переменные, процедуры и функции), а за ними следует тело самой подпрограммы.

Например, в разделе интерфейса объявлены:

```
procedure ISwap (var v1,v2: integer);
function IMax (v1,V2:integer);
```

В разделе реализации эти подпрограммы могут быть представлены в виде:

```
procedure ISwap;
var
Temp : integer;
begin
Temp := V1;
V1:= V2;
V2 := Temp;
end; {процедуры ISwap}

function IMax (v1,v2:integer):integer;
begin
if V1 > V2 then
IMax := V1
else
IMax := V2
end; {функции IMax}
```

Подпрограммы раздела реализации, которые неописаны в секции интерфейса, должны иметь полный заголовок procedure или function с именем и всеми параметрами.

Раздел инициализации

Раздел реализации модуля заключен между словами implementation и end. Но если внутри него присутствует слово begin, стоящее перед end и операторы между этими словами, то получивший-

ся составной оператор, похожий на тело главной программы, становится разделом инициализации модуля.

В этом разделе инициализируются структуры данных (переменных), используемые данным модулем или доступные другим программам, использующим этот модуль. Вы можете использовать этот раздел для открытия файлов, например, стандартный модуль Printer использует такой раздел для открытия на вывод текстового файла Lst. Файл Lst впоследствии можно использовать в программах, задавая его в операторах Write или Writeln.

При выполнении программы, использующей некоторый модуль, раздел инициализации вызывается перед выполнением тела главной программы. Если в программе используется несколько модулей, раздел инициализации каждого модуля вызывается (в порядке, указанном в операторе uses программы) до выполнения тела главной программы.

Использование модуля

Стандартные модули, которые использует главная программа, уже откомпилированы и хранятся в специальном машинном коде. Эти модули находятся в специальном файле TURBO.TPL и автоматически загружаются в память при работе системы Turbo Pascal.

В результате подключения модулей (как стандартных, так и пользовательских) к программе увеличивается (незначительно) время компиляции программы. Если модули загружаются из отдельных дисковых файлов (с дискет) может потребоваться дополнительное время из - за чтения с диска.

Для использования модулей необходимо, чтобы в начале программы присутствовало предложение uses, за которым следует список имен всех модулей, разделенных запятыми:

```
program Prog;
uses Unit1, Unit2, Unit3;
```

При компиляции этой информации к таблице символов программы прибавляется информации из раздела интерфейса модулей, а из раздела реализации добавляется машинный код модулей. Порядок описания модулей в предложении uses не имеет большого значения. Даже если Unit1 использует Unit2 можно объявить их в любом порядке. Компилятор сам определит, который из них должен следовать первым.

Если модуль Unit1 использует Unit2, а программа Prog не вызывает какие - либо подпрограммы в модуле Unit2 напрямую, то можно

"спрятать" подпрограммы в модуле Unit2, опуская его в операторе uses главной программы:

```
unit Unit1
uses Unit2
...
program Prog;
uses Unit1, Unit3;
...
```

В этом случае модуль Unit1 может вызвать подпрограмму (модуль) Unit2, а программа Prog вызывает подпрограммы Unit1 и Unit3, но не может непосредственно вызвать Unit2, т.к. эта подпрограмма не описана в его предложении uses.

Если предложение uses отсутствует, Turbo Pascal подсоединяет стандартный модуль System. Этот модуль обеспечивает выполнение некоторых стандартных подпрограмм и модулей Turbo Pascal.

Ссылки на описание модуля

Если вы включили модуль в свою программу, то все константы, типы данных, переменные, процедуры и функции, объявленные в интерфейсе этого модуля становятся доступными для вашей программы. Допустим, существует некоторый модуль:

```
unit MyStuff;
interface
const
MyValue := 915;
type
MyStars=(Deneb, Antares, Betelgeuse);

var
MyWord : string[20];
procedure SetMyWord(Star : MyStars);
function TheAnswer : integer;
implementation
...
end.
```

Часть модуля, которая описана в интерфейсе, доступна и может

быть использована в вашей программе. Поэтому, можно написать следующую программу:

```
program TestStuff;
uses MyStuff;

var
I : integer;
AStar : MyStars;
begin
Writeln(MyValue);
AStar: = Deneb;
SetMyWord(AStar);
Writeln(MyWord);
I: = TheAnswer;
Writeln(I);
end.
```

После включения предложения uses MyStuff в программу, появилась возможность ссылаться на все объявления и описания в секции интерфейса модуля MyStuff (MyWord, MyValue и т.д.).

Приведем еще одно использование этого модуля:

```
program TestStuff;
uses MyStuff;
const
MyValue: = 22;

var
I : integer;
AStar : MyStars;
function TheAnswer : integer;
begin
TheAnswer: = - 1;
end;
begin
Writeln(MyValue);
AStar: = Deneb;
SetMyWord(AStar);
Writeln(MyWord);
I: = TheAnswer;
```

```
Writeln(I);
end.
```

В этой программе переопределяются некоторые идентификаторы, объявленные в модуле MyStuff. В частности, программа будет использовать собственные описания для величин MyValue и TheAnswer, так как они были описаны позже, чем в модуле MyStuff. Если нужно использовать идентификаторы из MyStuff, то в этом случае перед каждым идентификатором помещается слово MyStuff с точкой (.).

Пример:

```
program TestStuff;
uses MyStuff;
const
MyValue = 22;

var
I : integer;
Astar : MyStars;
function TheAnswer : integer;
begin
TheAnswer: = - 1;
end;
begin
Writeln(MyStuff.MyValue);
Astar: = Deneb;
SetMyWord(AStar);
Writeln(MyWord);
I: = MyStuff.TheAnswer;
Writeln(I);
end.
```

Эта программа работает так же, как и первая, даже если MyValue и TheAnswer были переопределены. В действительности, первую программу можно было написать:

```
program TestStuff;
uses MyStuff;

var
I : integer;
```

```
AStar : MyStuff.MyStars;
begin
WriteIn(MyStuff.MyValue);
AStar: = MyStuff.Deneb;
MyStuff.SetMyWord(AStar);
WriteIn(MyStuff.MyWord);
I: = MyStuff.TheAnswer;
WriteIn(I);
end.
```

Заметим, что все идентификаторы, константы, типы данных, переменные или подпрограммы могут быть записаны с именем модуля (как показано выше). Тогда их значения будут использоваться именно из этого модуля, даже, если эти переменные были переопределены в основной программе.

Предложение USES раздела реализации

Как и в версии 5.0, Turbo Pascal 6.0 дает возможность использовать предложение uses в разделе реализации модулей. Это предложение должно немедленно следовать за ключевым словом implementation (реализация) так же, как и предложение uses в разделе интерфейса появляется сразу же за ключевым словом interface (интерфейс).

Предложение uses в разделе реализации позволяет сделать недоступными некоторые детали модуля, поскольку программы, используемые в разделе реализации, невидимы пользователям этого модуля. Однако более важно то, что это позволяет конструировать взаимно - зависимые модули. Поскольку в Turbo Pascal модули необязательно должны быть строго иерархическими, можно задавать циклические ссылки модулей.

Стандартные модули

Файл TURBO.TPL, который входит в состав программы Turbo Pascal содержит все стандартные модули (System, Overlay, Crt, Dos и Printer), кроме Graph и модулей совместимости Graph3 и Turbo3. Эти модули загружаются в память вместе с Turbo Pascal и всегда доступны для любой программы. Файл TURBO.TPL хранится на жестком диске в том же месте, что и сама программа TURBO.EXE (или TPC.EXE).

System

Модуль System содержит все стандартные и встроенные процедуры и функции системы Turbo Pascal. Этот модуль присоединяется к каждой создаваемой вами программе.

Dos

Модуль Dos включает многочисленные процедуры и функции Turbo Pascal, которые эквивалентны наиболее часто используемым вызовам операционной системы MS DOS (GetTime, SetTime, DiskSize и т.д.). Кроме того, здесь определяются две программы низкого уровня MsDos и Intr, которые позволяют использовать любой вызов MS DOS или системные прерывания, а также определяются некоторые другие константы и типы данных.

Crt

Модуль Crt обеспечивает набор специальных средств объявлений для ввода/вывода на PC - константы, переменные и программы. Их можно использовать для работы с экраном (работа с окнами, управление курсором, управление цветом). Есть возможность вводить с клавиатуры и управлять звуковым сигналом.

Printer

В модуле Printer объявляется переменная текстового файла LST, которая связывается с драйвером устройства, позволяя посылать стандартный вывод на принтер, используя операторы Write и Writeln. Например, включив модуль Printer в программу, можно написать следующее:

```
Write (Lst, 'The sum of', A:4, 'and', B:4, 'is');
c:=A+B;
Writeln (Lst, c:8);
```

Graph

Этот модуль не входит в файл TURBO.TPL, но должен находиться в том же вместе, где и вспомогательные файлы, расширения которых .BGI и .CHR. Поместите GRAPH.TPU в текущий каталог (где находит-

ся сама программа Turbo Pascal) или используйте справочник модулей для указания полного пути до GRAPH.TPU. (Если вы используете жесткий диск и программу Install для установки Turbo Pascal, то система уже установлена так, что можно использовать Graph).

Файл Graph это набор быстродействующих, эффективных графических подпрограмм, которые позволяют в полной мере использовать графические возможности PC. Этот модуль реализует независимый от устройства графический драйвер, поддерживающий различные графические адаптеры.

Turbo3 и Graph3

Эти модули предназначены только для совместимости новой версии 6.0 со старыми вариантами Паскаля. Turbo3 содержит переменные и несколько процедур, которые не поддерживаются Turbo Pascal 6.0, а Graph3 поддерживает полный набор графических программ версии 3.0.

Создание собственных модулей

Будем считать, что вы написали модуль IntLib, поместили его в файл INTLIB.PAS и откомпилировали. Результатом компиляции будет файл INTLIB.TPU. Для того чтобы можно было его использовать в программе, необходимо описать этот модуль в операторе uses.

Таким образом, программа может быть представлена в следующем виде:

```
program MyProg;
uses IntLib;
```

Заметим, что Turbo Pascal предполагает, что файл, в котором находится модуль, имеет такое же имя (до 8 символов), что и имя самого модуля. Если ваш модуль имеет имя MyUtilities, то Turbo Pascal будет искать файл с именем MYUTILIT.PAS.

Компиляция модулей

Модуль компилируется так же, как основная программа - создается при помощи редактора, а затем вызывается команда компиляции Compile / Compile (или Alt + F9). Но вместо файла с расширением .EXE, создается файл с расширением .TPU (модуль Turbo Pascal).

Можно оставить этот файл, как одиночный файл, а можно поместить его в библиотеку TURBO.TPL при помощи утилиты TPUMOVER.EXE (смотрите далее). В любом случае, вы можете поместить файл .TPU в справочник модулей, который задается в окне ввода Unit Directories (Options / Directories). Таким образом, вы можете ссылаться на эти файлы, когда они не находятся в текущем справочнике или в TURBO.TPL. (Эта команда позволяет указать несколько справочников для поиска модулей).

Чтобы найти модуль, указанный в операторе uses, компилятор вначале просматривает резидентные модули - модули, загруженные в память во время запуска компилятора Turbo Pascal из библиотеки TURBO.TPL. Если этого модуля нет среди резидентных, компилятор считает, что он должен быть на диске и иметь расширение .TPU. Компилятор ищет его вначале в текущем справочнике, а затем в справочниках, заданных командой O/D/Unit Directories или директивой /U в командной строке TPC.

Например, конструкция:

```
uses Memory;
```

где Memory не резидентный модуль заставляет компилятор искать MEMORY.TPU в текущем справочнике, а затем в каждом из справочников модулей.

Команды Compile/Make и Compile/Build (смотрите далее) компилируют модули, заданные в операторе uses, а исходные файлы ищутся так же, как .TPU файлы, причем имя исходного файла модуля принимается такое же, как имя модуля с расширением .PAS.

Напишем небольшой модуль, назовем его IntLib и поместим в него две простые подпрограммы - процедуру и функцию:

```
unit IntLib;
interface
procedure ISwap (var I, J : integer);
function IMax (I, J : integer) : integer;

implementation
procedure ISwap;
var
Temp : integer;
begin
Temp: = I;
```

```
I: = J;
J: = Temp;
end; {конец процедуры ISwap}

function IMax;
begin
if I > J then
IMax: = I
else
IMax: = J;
end;{ конец  функции IMax}
end.{ конец модуля IntLib}
```

Сохраним его в файле INTLIB.PAS и откомпилируем. Результирующий код помещается в файл INITLIB.TPU. Поместите его в справочник модулей, если он есть или оставьте в том же справочнике, где находится программа.

Напишем следующую программу, которая использует модуль IntLib:

```
program IntTest;
uses IntLib;

var
A, B : integer;
begin
Write ('Enter two integer values : ');
Readln (A, B);
ISwap (A, B);
Writeln ('A= ', A, 'B= ', B);
Writeln ('The max is ', IMax (A, B);

end. {Конец программы IntTest}
```

Эти примеры демонстрируют взаимодействие главной программы и некоторого модуля, который она использует.

Модули и большие программы

До сих пор мы использовали библиотеки модулей (набор стандартных программ) и отдельные подпрограммы, которые используются

несколькими программами. Другой случай использования модулей - построение больших программ. В Turbo Pascal существует два аспекта, позволяющих использовать модули:

1. Turbo Pascal имеет огромную скорость компиляции и редактирования.
2. Turbo Pascal может управлять несколькими файлами одновременно - главная программа и модули.

Обычно, большая программы делится на модули, которые группируют процедуры по их функциям (назначению). Может существовать глобальный модуль, который используется всеми другими модулями и главной программой. Он определяет глобальные константы, типы данных, переменные, процедуры и функции.

Приведем примерную схему большой программы:

```
program Editor
uses
Dos, Crt, Printer {стандартные модули из TURBO.TPL}
EditGlobals, {модули, написанные пользователем}
EditInit,
EditPrint,
EditRead,
EditWrite,
EditFormat;
{объявления программы, процедуры и функции}
begin {главная программа}
end.{конец программы Editor}
```

Заметим, что модули этой программы могут находиться в библиотеке TURBO.TPL или существовать, как отдельные .TPU файлы - в этом случае, Turbo Pascal будет сам управлять вашим проектом. Это значит, что при перекомпиляции программы Editor, компилятор Turbo Pascal проверит дату файлов .PAS и .TPU и перекомпилирует только те модули и файлы, которые были модифицированы.

Другая причина использования модулей в больших программах определяется ограничением размера кодовых сегментов. Обычно процессор компьютера ограничивают размер сегмента кода (части программы) до 64 Кб. Turbo Pascal позволяет снять эти ограничения, помещая каждый модуль в отдельный сегмент. Верхняя граница определяется памятью машины (PC - персональный компьютер) и операцион-

ной системой, то есть 640 Кб на большинстве PC. Без использования модулей размер программы ограничен до 64 Кб кода.

Утилита TPUMOVER

Допустим, что вам нужно добавить к стандартным модулям в файле TURBO.TPL, написанный и отлаженный вами модуль для того, чтобы он автоматически загружался в память при компиляции любой программы. Поместить его в библиотеку стандартных модулей Turbo Pascal можно при помощи утилиты TPUMOVER.EXE.

Кроме того, эта утилита используется для пересылки и удаления модулей из библиотеки с целью уменьшения размера файла TURBO.TPL и экономии памяти при его загрузке.

Теперь понятно, что писать собственные модули не очень сложно. Удачно сконструированный и реализованный модуль только упрощает разработку больших и сложных программ. Конкретная проблема решается только один раз, а не многократно для каждой программы.

УПРАВЛЕНИЕ ПРОЕКТОМ

Выше мы рассматривали, как писать программы на Turbo Pascal, как использовать стандартные модули и как писать свои собственные модули. Вы уже имеете понятие о том, что программы могут быть большими и разделяться на несколько файлов.

В этой главе мы рассмотрим, как объединить программу в модули, как использовать встроенные возможности Make и Build, как выполнять условную компиляцию внутри исходного файла и как оптимизировать код программы на скорость выполнения.

Организация программ

Turbo Pascal версии 6.0 позволяет разделить любую программу на кодовые сегменты (программные части). Главная программа после компиляции занимает один сегмент. Это значит, что она не может занимать в памяти больше 64 Кб. Однако, имеется возможность увеличить этот верхний предел, разбив программу на модули. Каждый модуль может содержать до 64 Кб машинных кодов при компиляции. Рассмотрим теперь, как организовать такую программу и как собрать ее в модули.

Первое действие, это объединить все глобальные определения - константы, типы данных, переменные в один модуль и назвать его, например, MyGlobals. Ваша программа может использовать модуль MyGlobals и обращаться ко всем глобальным объявлениям.

Второй возможный модуль назовем, например, MyUtils. В этом модуле можно собрать все подпрограммы (процедуры и функции), используемые в главной программе. Здесь должны быть собраны подпрограммы, которые не зависят от каких - либо других подпрограмм (за исключением других подпрограмм в самом модуле MyUtils).

Кроме этого, можно объединить процедуры и функции в логические группы. В некоторой группе можно определить несколько процедур и функций, которые наиболее часто используются главной программой, а затем создать группу процедур и функций, которые используются несколько реже. Подобные логические группы образуют прекрасный и функциональный модуль. Опишем теперь, как создать такой модуль:

1. Скопируйте все процедуры и функции в отдельный файл и удалите их из главной программы.
2. Откройте этот файл для редактирования.

3. Наберите следующие строки перед процедурами и функциями:

```
unit MyUtils;
intarface
uses MyGlobals
implementation
```

где MyUtils - имя вашего модуля (а так же имя редактируемого файла).

4. Наберите оператор end в конце файла.

5. Между interface и implementation скопируйте заголовки процедур и функций, вызываемых из главной программы. Заголовок это первая строка подпрограммы вместе со словами procedure или function. Если этот модуль использует другие модули, введите их имена, отделяя запятыми между словом MyGlobals и ";" в предложении uses.

6. Откомпилируйте этот файл.

7. Вернитесь в главную программу и добавьте имя этого модуля в предложение Uses (в данном случае это MyUtils).

Это идеальный вариант, если вы хотите организовать программу таким образом, чтобы она удобно модифицировалась, и перекомпилировалась, как можно быстрее. И что наиболее важно - такая оптимизация дает возможность работать с более компактными и легко управляемыми кусками (сегментами) программных кодов.

Инициализация

Напомним, что любой модуль может иметь свои собственные коды инициализации. Они выполняются автоматически при загрузке программы, и если программа использует различные модули, выполняется инициализационный код каждого из них. Порядок выполнения модулей определяется порядком их описания в предложении uses основной программы. Итак, если в программе описано:

```
uses MyGlobal, MyUtils, EditLib, GraphLib;
```

то секция инициализации MyGlobal будет вызвана первой, следующая - модуля MyUtils и т.д. Для создания раздела инициализации модуля поместите ключевое слово begin перед словом end в конце раздела реализации. Эти слова определяют раздел инициализации модуля аналогично тому, как begin ... end определяет главное тело программы,

процедуры или функции.

Вы можете поместить сюда (в раздел инициализации) любой код Паскаля. Здесь могут быть ссылки на все объявления этого модуля, как из интерфейсного раздела, так и из раздела реализации. Здесь так же могут быть ссылки к любым объявлениям интерфейсных частей всех модулей, используемых этим модулем.

Средства Build и Make

Turbo Pascal включает в себя очень важное и нужное средство управления проектом - встроенную утилиту Make. Рассмотрим ее значение в системе программирования.

Допустим, имеется программа MYAPP.PAS, которая использует четыре модуля: MyGlobals, MyUtils, EditLib, GraphLib. Эти четыре модуля - четыре текстовых файла MYGLOBAL.PAS, MYUTILS.PAS, EDITLIB.PAS, GRAPHLIB.PAS.

Далее, MyUtils использует модуль MyGlobals, а EditLib и GraphLib используют и MyGlobals, и MyUtils. При компиляции MYAPP.PAS компилятор ищет файлы MYGLOBAL.TPU, MYUTILS.TPU, EDITLIB.TPU и GRAPHLIB.TPU, загружает их в память, собирает их коды в файл MYAPP.PAS, компилирует и записывает их в файл MYAPP.EXE (если компилируется на диск).

Теперь допустим, что мы внесем изменения в модуль EDITLIB.PAS. Тогда для создания нового MYAPP.EXE необходимо перекомпилировать EDITLIB.PAS и MYAPP.PAS, т.е. всего два файла.

А если изменения внесены в секцию интерфейса модуля MYGLOBAL.PAS, то для создания новой версии MYAPP.EXE необходимо перекомпилировать уже все четыре модуля и сам файл MYAPP.PAS. Поэтому может показаться, что использование модулей не совсем выгодно (с точки зрения процедуры компиляции), однако не будем торопиться с выводами и рассмотрим возможности утилиты Make.

Make

В рассмотренном выше примере, Turbo Pascal предлагает решение - вы можете использовать опцию Make в меню Compile или нажать F9 (среды компилятора языка) и Turbo Pascal выполнит за вас всю работу. Процесс очень простой - после внесения изменений в какой - либо модуль или главную программу перекомпилировать надо только главную программу.

Компилятор Turbo Pascal осуществляет два основных вида проверок:

1. Во - первых, проверка даты и времени для каждого модуля, используемого программой. Дата сверяется у файлов с расширениями .PAS и .TPU. Если в файлы .PAS вносились изменения с тех пор, как был создан соответствующий .TPU файл, то этот файл .PAS перекомпилируется заново, создавая обновленный файл .TPU. Поэтому в первом примере (предыдущий параграф), когда изменения вносятся в EDITLIB.PAS, Turbo Pascal автоматически откомпилирует EDITLIB.PAS перед компиляцией MYAPP.PAS (при условии использования опции Make).
2. Вторая проверка - были ли внесены изменения в секцию интерфейса модифицируемого модуля. Если это имело место, то Turbo Pascal заново откомпилирует все модули, использующие данный модуль.
3. Во втором примере (предыдущий параграф) изменения были внесены в раздел интерфейса модуля MYGLOBAL.PAS. В этом случае Turbo Pascal перед компиляцией MYAPP.PAS автоматически перекомпилирует MYGLOBAL.PAS, MYUTIL.PAS, EDITLIB.PAS и GRAPHLIB.PAS (в описанном в разделе uses порядке).
4. Однако если вы модифицировали только раздел реализации, то перекомпиляция других зависимых модулей не требуется, поскольку (с их точки зрения) вы этот модуль не изменили (поскольку у них нет доступа в этот раздел).

При работе с компилятором командной строки, укажите опцию /M. Опция Make не воздействует на модули, находящиеся в TURBO.TPL.

Build

Опция Build, это частный случай утилиты Make. При использовании Build перекомпилируются все модули, используемые данной программой, исключая модули из библиотеки TURBO.TPL. Это более простой и надежный способ убедиться, что все будет полностью обновлено.

Для вызова Build из командной строки компилятора используйте опцию /B.

ИНТЕГРИРОВАННАЯ СРЕДЫ РАЗРАБОТКИ

Система Turbo Pascal - это больше, чем просто быстрый компилятор Паскаль - это эффективный компилятор Паскаль с интегрированной усовершенствованной средой разработки, легкой для изучения и использования (для краткости мы будем называть ее IDE). С Turbo Pascal не нужно использовать отдельные редактор, компилятор, редактор связей и отладчик (как для Fortran 77) для того, чтобы создавать, отлаживать и выполнять свои программы на Паскале. Все эти возможности встроены в оболочку Turbo Pascal, и все они доступны из IDE.

Вы можете начать построение своей первой программы на Turbo Pascal, используя редактор и компилятор, встроенные в IDE. В конце этой главы вы изучите усовершенствованную среду, напишете и сохраните небольшие программы, изучите некоторые основные навыки программирования. Встроенная контекстно - ориентированная справочная информация появляется на экране после нажатия некоторых клавиш (или отметки, щелчка мышкой). Можно получить справочную информацию (за исключением случаев, когда управление переходит к вашей программе) посредством нажатия клавиши F1. Меню Help (Alt + H) обеспечивает вас таблицей содержания системы справочной информации, подробным оглавлением, способностями поиска (Ctrl + F1), возможностью вернуться назад к другим экранам (Alt + F1) и подсказкой по справочной информации (повторное нажатие F1, если вы уже находитесь в системе справочной информации). Любой экран справочной информации может содержать одно или более ключевых слов (высвеченных, выделенных элементов), по которым можно получить дополнительную справочную информацию.

Компоненты

Существуют три видимых компоненты в интегрированной усовершенствованной среде:

1. Полоса меню в верхней части экрана.
2. Рабочая область окна в центре экрана.
3. Строка статуса внизу экрана.

Многие элементы меню предлагают также дополнительные диалоговые окна, позволяющие выполнять настройки самой среды разработки или управлять работой программы. Прежде чем мы рассмотрим каждый элемент меню в интегрированной среде, давайте опишем эти

наиболее общие компоненты. Далее мы будем иногда применять аббревиатуру для элементов меню. Например, для того чтобы выбрать добавление выражения для просмотра (Debug/Watch/Add Watch), мы будем говорить о выборе D/W/Add Watch.

Полоса меню и подменю

Полоса меню является основным способом доступа ко всем командам Главного меню. Полоса меню становится невидимой только в то время, когда вы просматриваете результаты вывода, полученные по своей программе. Если полоса меню активна, то заголовок меню будет высвечен (подсвечен) - это текущее выбранное меню. Если за командой какого - то меню следует знак многоточия (...), то выбор такой команды приведет к выводу на экран дополнительного диалогового окна. Если за командой следует стрелка (>), то команда ведет в другое меню. Команда без знака многоточия или стрелки указывает, что как только вы ее выбрали, произойдет какое - то действие.

Покажем теперь, как выбрать команды меню, используя только клавиатуру:

1. Нажмите клавишу F10 - это сделает полосу меню активной.
2. Чтобы выбрать меню, которое вы хотите посмотреть, используйте клавиши со стрелками влево - вправо, а для раскрытия меню нажмите клавишу Enter. Можно просто нажать высвеченную букву заголовка меню. Например, из полосы меню нажмите E, чтобы быстро показать состав меню Edit. Из любого места (даже при не активном меню) нажмите клавишу Alt и высвеченную (выделенную) букву в некотором меню для просмотра его содержимого. Чтобы прервать некоторое действие, нажмите клавишу Esc.

Можно также использовать мышку для выбора команд меню - этот процесс заключается в следующем:

1. Отметьте (щелкните мышкой) заголовок требуемого меню для его просмотра.
2. Отметьте (щелкните) требуемую команду открытого вами меню.

Вы можете также тащить мышку на заголовке меню к нужной команде данного меню. Освободите кнопку мышки на команде, которая нужна. (Если вы изменили свое решение - выйдите из меню и ни одна

команда не будет выбрана). Заметим, что некоторые команды меню являются недоступными, когда нет смысла их выбирать. Вы можете, однако, выбрать (высветить) недоступную команду, чтобы получить по ней подсказку. Turbo Pascal использует только левую кнопку мышки. Однако вы можете настроить ее правую кнопку, а также сделать некоторые другие настройки для мышки.

Сокращения

Система Turbo Pascal предлагает несколько быстрых способов для выбора команд меню. Например, пользователи мышки могут комбинировать двух - шаговый процесс в одно - шаговый, проведя мышкой от заголовка меню вниз к нужной команде меню и освобождая (отпуская) кнопку мышки для выбора требуемой команды.

Для клавиатуры можно использовать несколько быстрых методов (или горячих клавишей) для доступа к полосе меню и выбора команд. Быстрые методы для диалоговых окон работают так, как они сделаны в меню. При перемещении от окна ввода к группе кнопок или окон, вам нужно держать нажатой клавишу Alt и нажать высвеченную букву. Многие элементы меню имеют соответствующие горячие клавиши, одно - или двух - ключевые сокращения, которые немедленно активизируют эту команду или диалоговое окно. Можно также отметить (щелкнуть) мышкой сокращения в строке статуса. Следующая таблица перечисляет наиболее используемые в Turbo Pascal горячие клавиши.

Общие горячие клавиши.

Клавиша(и)	***Элемент меню***	***Функция***
F1	Help	Показывает экран подсказки.
F2	File/Save	Сохраняет файл, находящийся в активном окне редактора.
F3	File/Open	Появляется диалоговое окно и возможность открыть файл.
F4	Run/Go to Cursor	Запускает вашу программу до строки, на которой стоит курсор.
F5	Window/Zoom	Масштабирует активное окно.
F6	Window/Next	Проходит через все открытые окна.
F7	Run/Trace Into	Запускает вашу программу в режиме отладки с заходом внутрь процедур.

F8	Run/Step Over	Запускает вашу программу в режиме отладки, минуя вызовы процедур.
F9	Compile/Make	Делает Make текущего окна.
F10	None	Возвращает вас в полосу меню.

Горячие клавиши меню.

Клавиша(и)	***Элемент меню***	***Функция***
Alt+ПРОБЕЛ	Ё меню	Переносит вас в Ё (System) меню.
Alt+C	Compile меню	Переносит вас в Compile меню.
Alt+D	Debug меню	Переносит вас в Debug меню.
Alt+E	Edit меню	Переносит вас в Edit меню.
Alt+F	File меню	Переносит вас в File меню.
Alt+H	Help меню	Переносит вас в Help меню.
Alt+O	Options меню	Переносит вас в Options меню.
Alt+R	Run меню	Переносит вас в Run меню.
Alt+S	Search меню	Переносит вас в Search меню.
Alt+W	Window меню	Переносит вас в Window меню.
Alt+X	File/Exit	Завершает Turbo Pascal с выходом в DOS.

Горячие клавиши редактирования.

Клавиша(и)	***Элемент меню***	***Функция***
Ctrl+Del	Edit/Clear	Удаляет выбранный текст из окна, не помещая его в карман.
Ctrl+Ins	Edit/Copy	Копирует выбранный текст в карман.
Shift+Del	Edit/Cut	Помещает выбранный текст в карман и удаляет его.
Shift+Ins	Edit/Paste	Помещает текст из кармана в активное окно.
Ctrl+L	Search/Search Again	Повторяет последнюю команду Find или Replace.

Горячие клавиши управления окнами.

Клавиша(и)	***Элемент меню***	***Функция***
Alt+#	None	Показывает окно, где # - номер

		окна, которое вы хотите посмотреть.
Alt+0	Window/List	Показывает список открытых окон.
Alt+F3	Window/Close	Закрывает активное окно.
Alt+F5	Window/User Screen	Показывает экран пользователя.
Shift+F6	Window/Previous	Проходит назад через все открытые окна.
Ctrl+F5	Window/Size/Move	Изменяет размер или позицию активного окна.

Горячие клавиши встроенной справочной информации.

Клавиша(и)	***Элемент меню***	***Функция***
F1	Help/Contents	Открывает контекстно-ориентированный экран справочной информации.
F1 F1	Help/Help on Help	Вызывает справочную информацию по справочной информации (нужно нажать только F1, если вы уже находитесь в системе справочной информации).
Shift+F1	Help/Index	Вызывает оглавление справочной информации.
Alt+F1	Help/Previous Topic	Показывает предыдущий экран справочной информации.
Ctrl+F1	Help/Topic Search	Вызывает специфическую информацию по языку только в редакторе.

Горячие клавиши отладки/запуска.

Клавиша(и)	***Элемент меню***	***Функция***
Alt+F9	Compile /Compile	Компилирует последний файл в редакторе.
Ctrl+F2	Run/Program Reset	Переустанавливает выполняемую программу.
Ctrl+F4	Debug/Evaluate/ Modify	Вычисляет выражение.
Ctrl+F7	Debug/Add	Добавляет выражение для

	Watch	просмотра.
Ctrl+F8	Debug/Toggle BreakPoint	Устанавливает или очищает условные точки прерывания. Ctrl+F9 Run/Run запускает программу.

Окна Turbo Pascal

Почти все, что вы видите и делаете в среде Turbo Pascal, происходит в окнах. Окно - это область экрана, которую можно перемещать, изменять ее размеры, перекрывать, закрывать и открывать.

В Turbo Pascal вы можете иметь любое количество открытых окон (если это позволяет размер памяти), но в любой момент времени может быть активным только одно окно. Активное окно - это окно, с которым вы в настоящий момент времени работаете. Любая команда или текст, которую вы выбрали или набрали, относится только к активному окну. (Если один и тот же файл открыт в нескольких окнах, ваше действие будет применяться к файлу во всех этих окнах). Можно увеличить число окон, которые потенциально могут быть открыты, путем увеличения размера кучи, используя опцию Startup (Options / Environment).

Существуют несколько типов окон, но большинство из них имеют некоторые общие элементы:

- Полоса заголовка.
- Закрывающая кнопка.
- Полосы скроллинга.
- Уголок для изменения размеров окна.
- Кнопка масштабирования.
- Номер окна.

В среде Turbo Pascal можно отличить активное окно по двойной рамочке. Активное окно всегда имеет закрывающую кнопку, кнопку масштабирования, кнопки перемещения и уголок изменения размеров. Если ваши окна перекрываются, то активное окно всегда находится наверху остальных (на переднем плане).

Окно редактора в нижнем левом углу всегда показывает номера текущих строк и столбцов. Если вы изменили свой файл, то слева от номеров строки и столбца появится знак звездочки (*). Закрывающая кнопка окна - это кнопка в левом верхнем углу. Отметив (щелкнув мышкой) эту кнопку, вы можете быстро закрыть окно. (Можно также выбрать Window/Close или нажать Alt + F3). Окно справочной информации рассматривается, как временное и может быть закрыто посред-

ством нажатия Esc. Полоса заголовка - находящаяся выше всех горизонтальная строка окна, содержит имя и номер окна. Вы можете дважды отметить (щелкнуть) кнопку, находясь на полосе заголовка, для того чтобы масштабировать это окно. Можно также тащить мышкой за строку заголовка для перемещения окна на экране.

Каждое открытое окно в верхнем правом углу имеет свой номер, а клавиши Alt+0 выдают список всех открытых окон. Можно сделать окно активным (самым верхним) посредством нажатия Alt в комбинации с номером окна. Например, если окно справочной информации является #5, но скрыто под другими окнами, то для быстрого вынесения его на передний план можно нажать Alt+5.

Кнопка масштабирования окна всегда находится в верхнем правом углу. Если значок в этом углу изображает стрелку вверх, можно отметить эту стрелку для увеличения окна до максимально возможного размера. Если значок представляет собой двуглавую стрелку, то окно уже имеет свой максимальный размер. Если вы отметите двуглавую стрелку, то окно вернется к своему предыдущему размеру. Чтобы масштабировать окно с помощью клавиатуры, выберите Window/Zoom или нажмите клавишу F5.

Полосы скроллинга - это вертикальные или горизонтальные полосы. Вы можете использовать эти полосы для перемещения содержания окна. Щелкните мышкой стрелку на любом конце полосы скроллинга для перемещения на одну строчку. (Для непрерывного перемещения по тексту программы держите кнопку мышки нажатой). Отмечая, щелкая затененную область с любой стороны полосы скроллинга, вы можете перейти на целую страницу. Наконец, можно тащить полосу скроллинга "за любое место" для быстрого перемещения в любое место, соответствующее позиции полосы скроллинга. Полосы скроллинга позволяют пользователям мышки и клавиатуры видеть, в каком месте файла они находятся. Уголок изменения размеров находится в нижнем правом углу окна. Можно тащить мышкой за любой уголок, чтобы сделать окно больше или меньше. Вы можете отличить уголок изменения размеров окна по рамке из одной линии, вместо рамки из двойной линии, используемой в остальных местах окна. Для того чтобы изменить размеры с помощью клавиатуры, выберите Size/Move из Window меню, или нажмите Ctrl+F5.

Управление окнами

Следующая таблица содержит краткую информацию о том, как можно управлять окнами в Turbo Pascal. Заметим, что для выполнения этих действий мышка не нужна - можно использовать только клавиа-

туру.

Управление окнами.

Операция	***Способ***
Открыть Edit окно	Выберите File/Open для открытия файла и покажите его в окне или нажмите F3.
Открыть другие окна	Выберите требуемое окно из Window меню.
Закрыть окно	Выберите Close из Window меню (или нажмите Alt+F3), или щелкните закрывающую кнопку окна.
Активизировать окно	Щелкните в любом месте окна, или нажмите Alt плюс номер окна (в верхнем правом углу окна), или выберите Window/List или нажмите Alt+0 и выберите окно из списка, или выберите Window/Next или F6, чтобы сделать активным следующее окно (следующее в порядке, в каком они были открыты), или нажмите Alt+F6 для активизации предыдущего окна.
Передвинуть	Тащите его заголовок, или нажмите Ctrl+F5 для активного окна (Window / Size / Move) и используйте клавиши со стрелками, чтобы поместить окно там, где вам требуется, и нажмите Enter.
Изменить размер	Тащите уголок для изменения размера (или любой угол активного окна). Или выберите Window/Size/Move и нажимайте Shift одновременно с клавишами со стрелками, и нажмите Enter. Более быстрый способ состоит в нажатии Ctrl+F5 и затем одновременным использованием Shift и клавиш со стрелками.
Масштабировать	Щелкните кнопку масштабирования в верхнем правом углу активного окна, или дважды отметьте заголовок окна, или выберите Window/Zoom, или нажмите F5.

Строка статуса

Строка статуса появляется внизу экрана Turbo Pascal и выполняет следующие функции:

- Напоминает вам основные строки ключей и сокращений (или горячих клавиш), допустимых в этот момент в активном окне.
- Предоставляет вам быстрый вариант выполнения действий, отмечая горячие клавиши в строке статуса, вместо выбора команд из меню или нажатия последовательности клавиш.
- Говорит о том, какая функция выполняется в данный момент. Например, она показывает "Saving filename...", когда сохраняется редактируемый файл.
- Предлагает краткие советы по выбранной команде меню и элементам диалогового окна.

Как только вы переключили окна или изменили характер деятельности, строка статуса сразу же меняется. Одна из наиболее характерных строк статуса - это та, которую вы видите во время написания и редактирования программ в окне редактора.

Диалоговые окна

Если после команды меню следует многоточие (...), команда открывает диалоговое окно. Диалоговое окно - это наиболее удобный способ показать и установить многочисленные опции среды.

Когда вы делаете установки в диалоговых окнах, вы работаете с пятью основными типами управления экрана:

- Зависимые кнопки.
- Независимые кнопки.
- Кнопки действия.
- Окна ввода.
- Окна списка.

Если у вас цветной монитор, Turbo Pascal будет отмечать цветом различные элементы диалогового окна. Любое диалоговое окно имеет три стандартные кнопки: OK, Cancel, Help. При нажатии OK будут выполнены все сделанные вами выборы в этом окне, при выборе Cancel никаких изменений и действий сделано не будет, а диалоговое окно исчезнет. Клавиша Esc всегда является быстрым вариантом клавиатуры для Cancel (даже если Cancel кнопка не присутствует). Диалоговое окно Breakpoints Options является уникальным, поскольку оно не имеет кнопки Cancel.

Если вы используете мышку, то можете щелкнуть требуемую кнопку. Когда вы используете клавиатуру, то можете нажимать высве-

ченную букву элемента для его активизации. Например, нажатие "K" означает выбор кнопки OK. Нажимайте Tab или Shift + Tab для перехода в диалоговом окне от одного элемента к другому. Каждый элемент, становясь активным, высвечивается.

В любом диалоговом окне кнопка OK является кнопкой, заданной по умолчанию. Это означает, что достаточно нажать клавишу Enter для выбора этой кнопки. Помните о том, что переход к некоторой кнопке с помощью табуляции делает ее кнопкой по умолчанию.

Зависимые и независимые кнопки

Можно одновременно иметь любое число включенных независимых кнопок. Когда вы выбираете независимую кнопку, на ней появляется буква "X", указывающая, что кнопка находится в состоянии On (Включено). Пустая кнопка означает, что она находится в состоянии Off (Выключено). Можно управлять независимыми кнопками (устанавливать их в On), щелкая их или текст, содержащийся на них, или нажатием Tab до тех пор, пока кнопка не высветится, а затем нажать ПРОБЕЛ, или нажать высвеченную букву.

При выборе независимой кнопки в группе используйте клавиши со стрелками или высвеченную букву для выбора требуемого элемента, а затем нажмите ПРОБЕЛ для его окончательного выбора. На монохромных мониторах, Turbo Pascal указывает активную независимую кнопку или группу независимых кнопок посредством помещения символа шеврона (>>) следом за ней. При нажатии Tab шеврон перемещается к следующей группе независимых или зависимых кнопок.

Зависимые кнопки отличаются от независимых тем, что они предоставляют взаимно исключающие выборы. По этой же причине зависимые кнопки ВСЕГДА находятся в группе и только одна включенная кнопка (ни больше, ни меньше) может находиться в одной группе в один момент времени. Для выбора зависимой кнопки щелкните ее или находящийся на ней текст. При работе с клавиатурой нажмите высвеченную букву, или нажимайте Tab до тех пор, пока не высветится вся группа, а затем используйте клавиши со стрелками для выбора конкретной зависимой кнопки. После выбора зависимой кнопки нажмите Tab или Shift + Tab, для того чтобы выйти из группы.

Окна ввода и списки

Возможно, вы уже знакомы с окнами ввода - эти окна позволяют вам набирать текст. В окне ввода работают все основные клавиши ре-

дактирования текста (например, клавиши со стрелками, Home, End и переключение режима вставки/замены с помощью клавиши Ins). Если, достигнув нижнего края окна, вы продолжаете набирать текст, он автоматически передвигается вверх. Если текста больше, чем видно в строке окна, то в нижней части окна появляются значки стрелок (влево или вправо). Можно отмечать, щелкать мышкой эти значки для скроллинга - пролистывания текста. Если в окне ввода нужно набрать управляющие символы (такие как ^L или ^M - знак ^ обозначает клавишу Ctrl), то перед таким символом наберите ^P. Таким образом, например, набрав в окне ввода ^P^L, вы введете ^L.

Редактирование

Если вы давно пользуетесь продукцией фирмы Borland, то следующее краткое изложение новых свойств редактора поможет вам узнать те изменения, которые произошли в последней версии системы. Интегрированный редактор Turbo Pascal теперь имеет:

- Поддержку мышки.
- Поддержку больших файлов (до 1 Мб).
- Окна редактора, которые можно передвигать, перекрывать и изменять их размеры.
- Мультифайловые возможности, что позволяет открывать несколько файлов одновременно.
- Многочисленные окна, позволяющие иметь несколько представлений одного и того же файла или разных файлов.
- Разумный макроязык, позволяющий создавать свои собственные команды редактирования.
- Возможность брать текст или примеры из окна справочной информации.
- Редактируемый карман, допускающий вырезание, копирование и его передачу между окнами.

Пример программы

После загрузки Turbo Pascal (набрав TURBO и нажав Enter в ответ на подсказку DOS), вы увидите полосу меню, строку статуса, пустой экран и окно с информацией о версии продукта (выбор команды About из меню System, в любой момент времени приведет к появлению этой информации). При нажатии любой клавиши информация с версией исчезает, но среда с открытыми окнами остается.

Нажмите клавишу F10, чтобы войти в полосу меню, а затем F3 (сокращение для File/Open) для показа диалогового окна Open a File. Вы находитесь в окне ввода, поэтому наберите MYFIRST (необязательно набирать расширение .PAS, это предполагается "по умолчанию") и нажмите Enter. Теперь можно приступить к набору текста программы, нажимая Enter в конце каждой строки:

```
program MyFirst;
var
A,B: Integer;
Ratio: Real;
begin
Write('Enter two numbers: ');
Readln(A,B);
Ratio := A/B;
Writeln('The ratio is ',Ratio);
Write('Press <Enter>...');
Readln;
end.
```

Не забывайте о точке с запятой в конце каждой строки, а за последним оператором end поставьте просто точку. Для удаления символов используйте клавишу Backspace, а для передвижения внутри окна редактора - клавиши со стрелками управления курсором.

Поскольку вы набираете и запускаете эту программу, не зная, что она делает, мы приведем здесь некоторые пояснения. Первая введенная строка задает имя программе MyFirst. Это утверждение - оператор необязательно, но неплохо включать его в программу. Следующие три строки со словом var объявляют несколько переменных. Переменные A и B имеют тип Integer (целое число), то есть, они могут содержать целые числа, такие как 52, -421, 0, 32, 283 и т.д. Переменная Ratio объявлена, как Real (действительное число), это означает, что она может хранить дробные числа, такие как 423.328 или -0.032.

Остальная часть программы содержит выполняемые утверждения - операторы, а слово begin говорит о начале программы. Утверждения (операторы программы) разделяются точкой с запятой и содержат инструкции записи на экран (Write и Writeln), чтения с клавиатуры (Readln) и выполнения вычислений (R: = A/B). Оператор Readln в конце программы приводит к остановке ее выполнения (пока вы не нажмете Enter), так что вы сможете исследовать результаты вывода программы. Выполнение программы начинается с первой инструкции по-

сле begin и продолжается до тех пор, пока не встретится end.

После набора этой программы нужно сохранить ее в постоянной памяти на жестком диске. Для этого выберите команду Save из меню File, нажав F10, а затем F для появления меню File и S для выбора команды Save. Наиболее легкий способ сохранения программы - использовать сокращение для File/Save, нажав клавишу F2.

Компиляция программы

Для компиляции программы выберите опцию Compile в основном меню системы. Можно нажать F10 и C, а комбинация Alt + C приведет вас прямо к этой опции. Другая комбинация клавиш Alt + F9 - это самый быстрый путь начать компиляцию.

Turbo Pascal скомпилирует вашу программу, переводя ее с языка Паскаль (который можно читать) на машинный код для микропроцессора (который может выполнить ваша PC). Вы не увидите машинный код, он хранится в оперативной памяти (или на диске).

Как и английский язык, Паскаль имеет грамматические правила, которые нужно выполнять. Однако в отличие от английского языка, структура языка Паскаль является нетерпимой к сленгу или плохому синтаксису - компилятор всегда должен понимать, ЧТО вы хотите сказать. В языке Паскаль, если вы используете неверные слова или символы или неправильно организовали всю программу, получите ошибку времени компиляции (синтаксическую).

Наиболее возможной ошибкой для начинающего программиста на языке Паскаль будет:

Unknown identifier - неизвестный идентификатор

или

';' expected - ожидается ';'

Паскаль требует, чтобы вы объявили все переменные, типы данных, константы и подпрограммы, т.е. все идентификаторы программы перед их использованием. Если вы обратитесь к необъявленному идентификатору или пропустите его, то получите ошибку. Другими распространенными ошибками являются несоответствие пар begin . . end, присваивание несовместимым типам данных (например, присваивание действительного числа целому), неверное число и тип параметров в вызовах процедур и функций и т.д.

Когда вы начинаете компиляцию, в центре экрана появляется окно, содержащее информацию о компиляции данной программы. Если во время компиляции не произошло никаких ошибок, то в этом окне появится сообщение "Compilation successful: press any key" - компиляция успешна: нажмите любую клавишу. Это окно останется на экране до тех пор, пока вы не нажмете любую клавишу.

Если во время компиляции произошла ошибка, Turbo Pascal останавливается, устанавливает курсор на ошибку в редакторе и показывает сообщение об ошибке внизу окна редактора. (Первое нажатие клавиши очистит это сообщение, а Ctrl + Q, W будет показывать его снова до тех пор, пока вы не измените свой файл или не перекомпилируете его). Сделайте исправления, сохраните обновленный файл и компилируйте его снова.

Выполнение программы

После фиксации ошибок, переключитесь в основное меню системы и выберите Run/Run (или нажмите Ctrl + F9). Вы попадете на экран пользователя, и на этом экране появится сообщение:

```
Enter two numbers:
```

Наберите два любых целых числа с пробелом между ними и нажмите Enter. Появится следующее сообщение:

```
The ratio is
```

а за ним - отношение первого числа ко второму. В следующей строке появится сообщение "Press <Enter>..." и программа будет ждать нажатия клавиши Enter. Чтобы посмотреть вывод (результат работы) своей программы, выберите Window/User Screen (или нажмите Alt + F5).

Если во время выполнения программы произошла ошибка, то на экране появится сообщение, которое выглядит следующим образом:

```
Run-time error <errnum> at <segment>:<offset>
```

где <errnum> - это соответствующий номер ошибки, а :<offset> - адрес в памяти, где произошла ошибка. После вывода сообщения об ошибке вы окажетесь в точке расположения ошибки в своей программе, с описательным сообщением об ошибке,

показанным в строке статуса редактора. Пока сообщение находится в строке статуса редактора, можно нажать F1 для получения справочной информации по конкретной ошибке. Нажатие любой другой клавиши приводит к исчезновению сообщения об ошибке. Если вам нужно будет найти местоположение ошибки снова, выберите Search/Find Error.

Когда ваша программа закончит выполнение, вы вернетесь в то место программы, с которого начинали. Теперь вы можете модифицировать программу, если хотите. Если вы выберете команду Run/Run перед внесением изменений в свою программу, Turbo Pascal немедленно выполнит ее снова без перекомпиляции.

Как только вы вернетесь назад в IDE после выполнения своей программы, вы можете просмотреть вывод своей программы посредством выбора команды Run/User Screen (или нажатия Alt + F5).

Вторая программа

Теперь мы будем писать вторую программу, которая построена на основе первой. Если вы вышли из Turbo Pascal, используя команду DOS Shell из меню File, вы можете вернуться в среду Turbo Pascal, набрав Exit в ответ на подсказку DOS. Если вы вышли, используя Exit из меню File, вы можете набрать:

```
TURBO MYFIRST.PAS
```

для того чтобы вернуться в IDE. Это перенесет вас прямо в редактор. Теперь измените программу MYFIRST.PAS следующим образом:

```
program MySecond;
var
A,B: Integer;
Ratio: Real;
begin
repeat
Write('Enter two numbers: ');
Readln(A,B);
Ratio := A/B;
Writeln('The ratio is ',Ratio:8:2);
Write('Press <Enter>...');
Readln;
until B = 0;
end.
```

Вам нужно сохранить это, как отдельную программу, поэтому войдите в меню File, выберите Save As и наберите MYSECOND.PAS, а затем нажмите Enter. Компилируйте и запускайте свою вторую программу, используя Ctrl + F9. Поскольку вы сделали изменения в программе, Turbo Pascal автоматически откомпилирует программу перед ее запуском.

В этой новой программе были сделаны существенные изменения. Утверждения (операторы) были заключены в цикл repeat . . until. Это приведет к тому, что все утверждения между repeat и until будут выполняться до тех пор, пока выражение, следующее за until, не станет True (Истинно). Это выражение проверяет, равно ли значение B нулю или нет. Если B имеет значение 0, цикл должен завершиться.

Запустите свою программу, проверьте некоторые значения, а затем введите 1 0 и нажмите Enter. Ваша программа прекратит работу, но не совсем так, как вы предполагали - она завершится с ошибкой времени выполнения, и вы вернетесь обратно в редактор, а курсор будет стоять на строке:

```
Ratio: = A/B;
```

Вверху окна редактора появится сообщение:

```
Error 200: Division by zero - ошибка 200: деление на 0.
```

Отладка программы

Если вы программировали раньше, то можете знать эту ошибку и методы ее фиксации. Но давайте воспользуемся этой возможностью, чтобы показать, как использовать интегрированный отладчик, встроенный внутрь системы Turbo Pascal 6.0. Интегрированный отладчик Turbo Pascal позволит вам передвигаться по коду программы по строкам. В то же время, вы можете просматривать переменные и следить, как изменяются их значения.

Чтобы начать сеанс отладки, выберите команду Run/Trace Into (или нажмите F7). Если ваша программа нуждается в перекомпиляции, Turbo Pascal сделает это. Первое утверждение (в данном случае это begin) в теле программы будет высвечено, и с этого момента мы будем называть эту высвеченную полосу - полосой запуска.

Первое нажатие клавиши F7 инициализирует сеанс отладки. Теперь нажмите F7 снова, чтобы начать выполнение программы - отладчик выполнит невидимый код запуска. Нажмите F7 еще раз - появится

экран пользователя. Это произойдет потому, что утверждение Readln ожидает ввода двух чисел. Наберите два целых числа, разделенные пробелом, и убедитесь, что второе число - не нуль. Теперь нажмите Enter - вы вернетесь назад в окно редактора, с полосой запуска на утверждении присваивания в строке 9.

Нажмите F7 и выполните утверждение присваивания. Теперь полоса запуска находится на утверждении Writeln в строке 10. Нажмите F7 дважды - теперь вы должны выполнить Readln в строке 12. Снова нажмите F7, посмотрите вывод своей программы, а затем нажмите Enter. Полоса запуска теперь находится на предложении until - нажмите F7 несколько раз, и вы вернетесь к началу цикла repeat.

Если вы в качестве второго числа наберете ноль, то при очередном нажатии клавиши F7, в момент присваивания A/B переменной Ratio, произойдет аварийное завершение программы, и вверху окна редактора опять появится сообщение об ошибке "Division by zero".

Теперь вы поняли, каким мощным средством является отладчик. Можно передвигаться в программе строка - за - строкой, можно показывать значение переменных и выражений своей программы, просматривать изменение их значений по мере выполнения программы.

Фиксирование программы

Теперь возможно вы задумались о том, что неправильно в вашей программе - если вы вводите значение 0 для второго числа, программа завершается с ошибкой времени выполнения.

Для того чтобы исправить эту ошибку нужно не делите "A" на "B", если "B" имеет значение 0. Отредактируйте свою программу так, чтобы она выглядела следующим образом:

```
program MySecond;
var
A,B: Integer;
Ratio: Real;
begin
repeat
Write('Enter two numbers: ');
Readln(A,B);
if B = 0 then
Writeln('The ratio is undefined')
else
begin
```

```
Ratio := A/B;
Writeln('The ratio is ',Ratio:8:2);
end;
Write('Press <Enter>...');
Readln;
until B = 0;
end.
```

Теперь запустите эту программу (или сами, или используя отладчик). Программа сама остановится после сообщения "The ratio is undefined. Press <Enter>..." - отношение не определено. Нажмите <Enter>... и аварийной остановки теперь не будет.

СИСТЕМА DELPHI

Среда разработки прикладных программ Delphi - это комбинация нескольких новых и важнейших технологий:

- Высокопроизводительный компилятор (который выполняет перевод символов и операторов некоторого алгоритмического языка на язык понятный компьютеру) программы, написанной на алгоритмическом языке (в данном случае это язык Pascal) в машинный код.
- Объектно - ориентированная модель компонент системы - модель, которая привязывает алгоритмический язык к визуальным (т.е. находящимся на экране) компонентам среды разработки.
- Визуальное и очень быстрое построение приложений (компьютерных программ) из программных прототипов - готовых шаблонов.

Рассмотрим теперь эти составляющие системы Delphi более подробно.

Компилятор в машинный код

Компилятор, встроенный в Delphi, обеспечивает высокую производительность, необходимую для построения различных приложений (прикладных компьютерных программ). Этот компилятор в настоящее время является самым быстрым в мире, и его скорость достигает 120 тысяч строк в минуту на компьютере с процессором Intel 486DX33. Он предлагает легкость разработки и быструю проверку готового программного блока, характерного для языков четвертого поколения (4GL). Кроме того, Delphi обеспечивает быструю разработку многих приложений вообще без необходимости писать вставки на алгоритмическом языке или ручного написания кода программы (хотя это вполне возможно).

В процессе построения приложения разработчик выбирает из палитры компонентов (шаблонов) готовые, входящие в состав системы компоненты. Еще до компиляции он видит результаты своей работы, а после подключения к источнику данных их можно видеть отображенными на Форме, которая позволяет перемещаться по данным, представлять их в том или ином виде. В этом смысле проектирование в Delphi мало, чем отличается от проектирования в интерпретирующей среде, однако после выполнения компиляции мы получаем код программы, который исполняется в 10 - 20 раз быстрее, чем то - же самое,

сделанное при помощи интерпретатора. Кроме того, в Delphi компиляция производится непосредственно в машинный код, в то время как существующие компиляторы превращают программу в так называемый p - код, который затем интерпретируется виртуальной p - машиной. Это не может не сказаться на фактическом быстродействии готового приложения.

Объектно - ориентированная модель

Основной упор объектно - ориентированной модели в Delphi делается на максимальном использовании готового кода блоков программы. Это позволяет разработчикам быстро строить приложения из заранее подготовленных объектов, а также дает им возможность создавать свои собственные объекты для среды Delphi. Все в Delphi написано на Delphi, поэтому разработчики имеют доступ к тем же объектам и инструментам, которые использовались для создания самой среды разработки. Поэтому нет никакой разницы между объектами, поставляемыми фирмой Borland или третьими фирмами и объектами, которые вы можете создать сами.

В стандартную поставку Delphi входят основные объекты, которые образуют удачно подобранную иерархию из более трех сотен базовых классов. Но если возникнет необходимость в решении, какой - то специфической проблемы на Delphi, то прежде чем пытаться решать проблему "с нуля", просмотрите список свободно распространяемых или коммерческих компонент, разработанных третьими фирмами, количество которых в настоящее время превышает несколько сотен.

Быстрая разработка приложения

Среда Delphi включает в себя полный набор визуальных инструментов (объектов) для скоростной разработки приложений (RAD - Rapid Application Development), поддерживающий разработку пользовательского интерфейса. VCL - библиотека визуальных компонентов, включает в себя стандартные объекты построения пользовательского интерфейса, объекты управления данными, графические объекты, объекты мультимедиа, диалоги и объекты управления файлами.

В Delphi визуальные компоненты пишутся на объектном Паскале (Object Pascal), на том же Паскале, на котором пишется алгоритмическая часть приложения. Поэтому все визуальные компоненты Delphi получаются открытыми для надстройки и переписывания.

СОСТАВ И НАЗНАЧЕНИЕ DELPHI

В настоящее время выпущено две версии Delphi - одна (Delphi Client/Server) адресована для разработчиков компьютерных приложений в архитектуре "клиент - сервер", а другая (Delphi for Windows) предназначена для создания локальных приложений на отдельно взятом компьютере.

Клиент - серверная версия Delphi

Клиент - серверная версия Delphi (Client/Server Edition) адресована корпоративным разработчикам (большим компаниям с собственной локальной компьютерной сетью), желающим разрабатывать высокопроизводительные приложения для рабочих групп и корпоративного (коллективного) применения.

Клиент - серверная версия включает в себя следующие блоки:

- SQL Links - специально написанные драйверы для доступа к базам данных Oracle, Sybase, Informix, InterBase.
- Локальный сервер InterBase SQL - сервер для работы в Windows.
- ReportSmith Client/server Edition - генератор отчетов для SQL - серверов.
- Visual Query Builder - средство визуального построения SQL - запросов.

Delphi for Windows

Delphi for Windows предназначен для разработчиков высокопроизводительных персональных приложений и предлагает такую же среду для быстрой разработки и первоклассный компилятор, как и клиент - серверная версия. Эта среда позволяет разработчику быстро изготавливать персональные приложения, работающие с персональными базами данных типа dBase и Paradox. В Delphi for Windows, как и в Delphi Client - Server, входят:

- Компилятор Object Pascal (этот язык является расширением языка Borland Pascal 7.0).
- Генератор отчетов ReportSmith 2.5 (у которого, правда, отсут-

ствует возможность работы с SQL - серверами).

- Среда визуального построителя программных приложений.
- Библиотека визуальных компонент VCL.
- Локальный сервер InterBase.

Назначение Delphi

В первую очередь, среда Delphi предназначена для профессионалов - разработчиков корпоративных информационных систем. Не секрет, что некоторые удачные продукты различных фирм, предназначенные для скоростной разработки приложений (RAD), прекрасно работают при изготовлении достаточно простых приложений, однако, разработчик сталкивается с непредвиденными сложностями, когда пытается сделать что - то действительно сложное.

В системе Delphi такие ограничения просто отсутствуют. Хорошее доказательство тому, это тот факт, что сам Delphi разработан на Delphi. Однако система Delphi предназначена не только для программистов - профессионалов. Любой программист на Паскале способен практически сразу профессионально освоить Delphi. Специалисту, ранее использовавшему другие программные продукты, придется труднее, однако самое первое работающее приложение он сможет написать в течение первого же дня работы на Delphi.

Особенности Delphi

В отличие от большинства Паскаль - компиляторов, транслирующих программу в p - код, в Delphi программный текст компилируется непосредственно в машинный код. В результате этого Delphi - приложения исполняются в 10 - 20 раз быстрее, особенно приложения, использующие математические функции.

Готовое программное приложение может быть изготовлено либо в виде исполняемого модуля (исполняемого EXE файла), либо в виде динамической библиотеки, которую можно использовать в приложениях, написанных на других языках программирования.

Открытая архитектура

Благодаря открытой архитектуре, приложения, изготовленные при помощи Delphi, работают надежно и устойчиво, а Delphi поддерживает использование уже существующих объектов. Из готовых, входящих в состав Delphi компонент, работающие приложения собираются очень

быстро. Кроме того, поскольку Delphi имеет полностью объектную ориентацию, разработчики могут создавать свои собственные повторно используемые объекты.

Delphi предлагает разработчикам, как в составе команды, так и индивидуальным, открытую архитектуру, позволяющую добавлять компоненты, где бы они ни были изготовлены, и оперировать этими вновь введенными компонентами в визуальном построителе.

Two - way tools - однозначное соответствие между визуальным проектированием и классическим написанием текста программы. Это означает, что разработчик всегда может видеть код программы, соответствующий тому, что он построил при помощи визуальных инструментов и наоборот.

Визуальный построитель интерфейсов (Visual User Interface Builder) дает возможность быстро создавать клиент - серверные приложения визуально, просто выбирая компоненты из соответствующей палитры компонент.

СТРУКТУРА СИСТЕМЫ DELPHI

После запуска системы Delphi на экране появляется ее основное рабочее окно, показанное на рис.1. В самом верху экрана находится Главное меню со словами File, Edit и т.д. Ниже Главного меню располагается Панель инструментов, которая поделена на две части. Справа, горизонтально располагаются палитры компонент (Standard, Additional и т.д. - это первая часть Панели инструментов). Щелкнув мышкой по имени палитры, мы выводим на экран ее панель, на которой располагаются кнопки с изображениями - иконками. Если курсор задерживается на одной из иконок, под ней в желтом прямоугольнике появляется подсказка (рис.1).

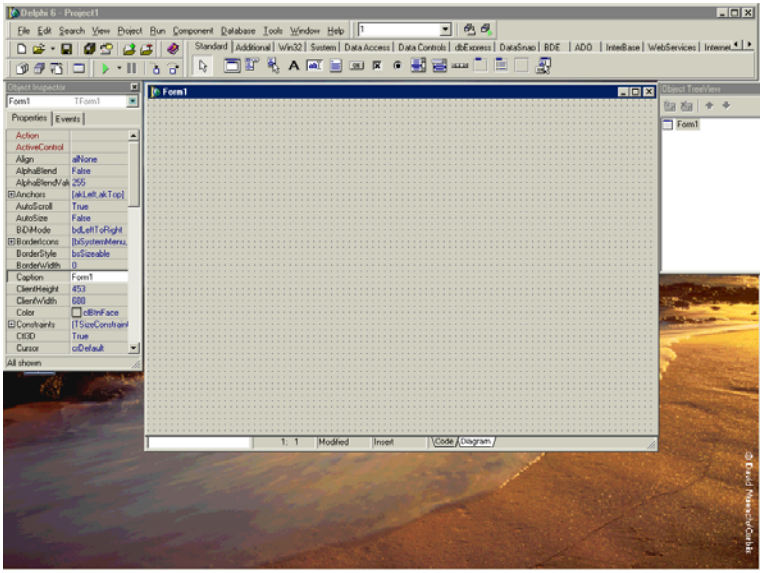

Рис.1. Общий вид окна Delphi.

Слева вверху окна системы находятся кнопки управления проектом в целом, которые во многом совпадают со стандартными кнопками, которые используются, например, в Windows или Word (Это вторая часть Панели инструментов).

Из палитры компонент можно выбирать компоненты (щелкнув по ним мышкой, а затем в окне Формы), из которых строится любое приложение. Компоненты включают в себя, как визуальные, так и логиче-

ские структуры. Такие объекты, как кнопки, поля и окна редактирования - это визуальные компоненты или структуры, а таблицы, отчеты и т.д. - это логические компоненты.

Ниже панелей инструментов (кнопки управления и палитра компонент) располагается окно Формы (Form, в данном случае под номером 1 - см. рис.1), а левее ее находится панель свойств, выделенного на Форме объекта (Object Inspector). За окном Формы находится окно редактора (рис.2), который позволяет просматривать и писать код программы для любого выделенного на Форме объекта. Переход между окнами Формы и редактора выполняется щелчком мышки по видимой части соответствующего окна.

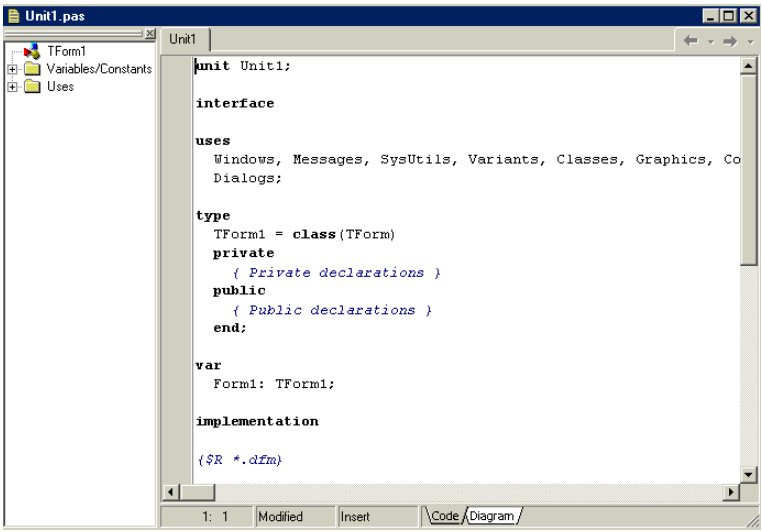

Рис.2. Окно редактора.

Поскольку в Delphi мы строим свою программу визуальным образом, все компоненты имеют свое графическое представление в поле Формы, чтобы можно было оперировать ими соответствующим образом. Но для готовой, работающей программы видимыми остаются только визуальные компоненты.

Компоненты сгруппированы на страницах палитры компонент по своим функциям. Например, компоненты, представляющие Windows "Common dialogs" размещены на странице палитры с названием Dialogs.

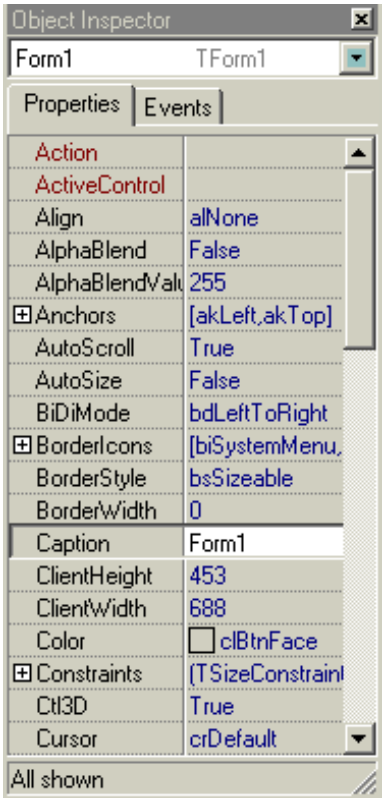

Рис.3. Окно свойств объектов.

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

В среде Delphi вы можете определить свою группу компонент или объектов и поместить ее на отдельной странице палитры, а если возникнет необходимость, то можно перегруппировать компоненты или удалить неиспользуемые.

Интеллектуальный редактор

Редактирование ваших программ можно осуществлять, используя запись и исполнение команд, работу с текстовыми блоками, настраиваемые комбинации клавиш и цветовое выделение строк, применяя для этого редактор системы Delphi, окно которого показано на рис.2.

Каждый блок программы можно открыть в своем, отдельном окне такого редактора, просмотреть код программы, внести, если нужно, необходимые изменения и сохранить их на жестком диске компьютера.

Система Delphi обладает мощнейшим, встроенным в редактор графическим отладчиком, позволяющим находить и устранять ошибки в коде программы. Вы можете установить точки остановки работы программы, проверить и изменить любые переменные, при помощи пошагового выполнения программы в точности понять ее поведение.

Инспектор объектов

Инспектор объектов, который приведен на рис.3, представляет собой отдельное окно, где вы можете в период проектирования програм-

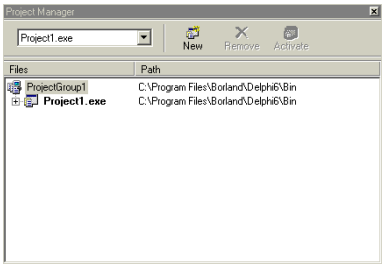

Рис.4. Окно менеджера проектов.

мы устанавливать значения свойств и событий любых объектов (Properties & Events), которые используются в вашей программе.

Менеджер проектов

Менеджер дает возможность разработчику просмотреть все модули в соответствующем проекте и снабжает вас удобным механизмом для управления проектами в целом (рис.4). Менеджер проектов показывает имена файлов, время и дату выбранных форм и т.д. Можно немедленно попасть в текст программы или на Форму проекта, просто щелкнув мышкой по соответствующему имени.

Навигатор объектов

Показывает библиотеку доступных объектов и осуществляет навигацию по вашему приложению (рис.5). Можно посмотреть иерархию объектов, перекомпилированные модули в библиотеке, список глобальных имен вашего кода.

Эксперты

Это набор инструментальных программ, облегчающих проектирование и настройку ваших приложений. Имеется дополнительная возможность подключать эксперты, разработанные самостоятельно. Потенциально это та возможность, при помощи которой третьи фирмы могут расширять Delphi - инструменты. Набор экспертов включает в себя:

- Эксперты Формы, работающие со многими базами данных.
- Эксперт стилей и шаблонов пользовательских приложений.
- Эксперт шаблонов Форм.

В состав дополнительного приложения RAD Pack входит эксперт для преобразования ресурсов, изготовленных для Borland Pascal 7.0 в Формы Delphi.

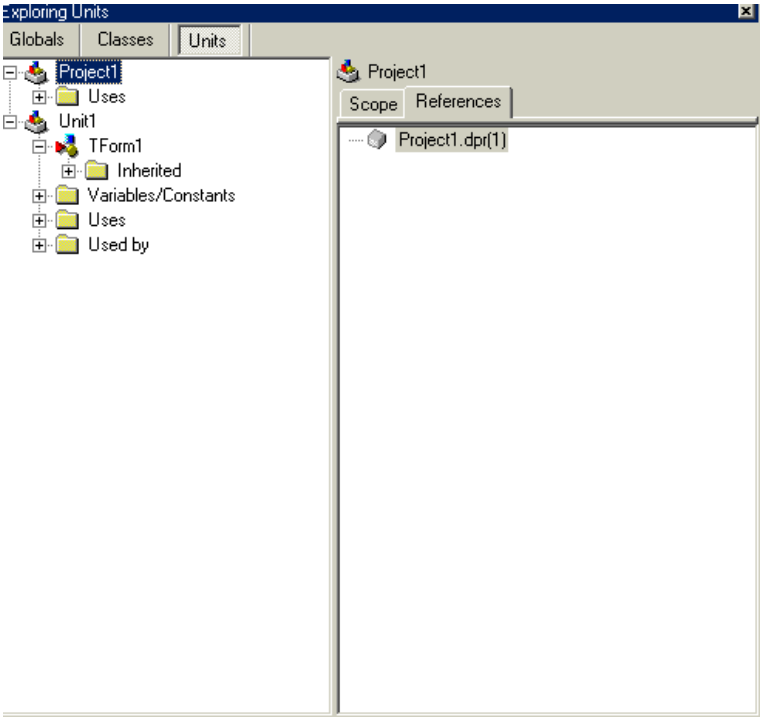

Рис.5. Окно навигатора объектов.

Библиотека Визуальных Компонент

Компоненты, используемые при разработке приложений в Delphi (а также самим Delphi), встроены в среду разработки приложений и представляют собой, набор типов объектов, используемых в качестве фундамента при строительстве любого приложения.

Этот костяк называется Visual Component Library (VCL - библиотека визуальных компонент). В VCL есть такие стандартные элементы управления, как строки редактирования, статические элементы управления, строки редактирования со списками, списки объектов и т.д. Имеются и такие компоненты, которые ранее были доступны только в библиотеках третьих фирм - табличные элементы управления, закладки, многостраничные записные книжки.

VCL содержит специальный объект, предоставляющий интерфейс графических устройств Windows и позволяющий разработчикам рисо-

вать, не заботясь об обычных для программирования в среде Windows деталях.

Ключевой особенностью Delphi является возможность не только использовать готовые визуальные компоненты для строительства приложений, но и создавать новые компоненты. Такая возможность позволяет разработчикам не переходить в другую среду разработки, а наоборот, встраивать новые инструменты в существующую среду. Кроме того, можно улучшить или полностью заменить, существующие "по умолчанию" компоненты Delphi.

Формы, модули и метод разработки

Формы это объекты, на которые вы помещаете другие объекты, (компоненты) которые в свою очередь, используются для создания пользовательского интерфейса вашего приложения. Модули состоят из кода программы, который реализует функционирование вашего приложения, обработчика событий для Форм и их компонент.

Информация о Формах хранится в двух типах файлов - .dfm и .pas, причем первый тип файла - двоичный, который хранит образ Формы и ее свойства, а второй тип описывает функционирование обработчиков событий и поведение компонент. Оба файла автоматически синхронизируются Delphi, так что если добавить новую Форму в ваш проект, связанный с ним файл .pas будет создан автоматически и его имя будет добавлено в ваш проект.

Такая синхронизация и делает Delphi two - way - инструментом, обеспечивая полное соответствие между кодом программы и ее визуальным представлением. Как только вы добавите новый объект или код, Delphi устанавливает, так называемую, "кодовую синхронизацию" между визуальными элементами и соответствующими им кодовыми представлениями.

Например, предположим, что вы добавили описание поведения Формы, чтобы показывать окно сообщения по нажатию кнопки. Такое описание появляется, если дважды щелкнуть мышкой непосредственно на объект Button (Кнопка) на Форме или дважды щелкнуть мышью на строчке OnClick (По щелчку) на странице Events (События) в Инспекторе объектов (Object Inspector).

В любом случае Delphi создаст процедуру или заголовок метода, куда вы можете добавить код:

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
end;
```

Создавая этот код, система Delphi автоматически формирует декларацию объекта TForm1, которая содержит процедуру ButtonClick, представляющую собой обработчик события:

```
TForm1 = class (TForm)
Button1: Tbutton;

procedure Button1Click(Sender: TObject);

private
{ Private declarations }

public
{ Public declarations }
end;
```

Конечно, после получения этого кода вы можете решить, что автоматически созданные имена вас не устраивают, и заменить их любыми другими. Например, имя Button1 можно заменить на Warning. Это можно сделать, изменив свойство Name для Button1 при помощи Инспектора объектов. Как только вы нажмете клавишу Enter, система Delphi автоматически произведет соответствующую синхронизацию в коде программы. Так как объект TForm1 существует в самом коде, вы свободно можете добавлять любые другие поля, процедуры, функции или Object definition (Определения объектов). Например, вы можете вручную дописать в коде программы свою собственную процедуру, обрабатывающую событие, а не делать это визуальным методом.

Следующий пример показывает, как это можно сделать. Обработчик принимает аргумент типа TObject, который позволяет нам определить, если необходимо, кто инициировал событие. Это полезно в случае, когда несколько кнопок вызывают общую процедуру обработки события:

```
TForm1 = class(TForm)
Warning: TButton;
Button1: TButton;

procedure WarningClick(Sender: TObject);
```

```
procedure NewHandler(Sender: TObject);
private
{ Private declarations }

public
{ Public declarations }
end;
```

Визуальная среда сама распознает, что новая процедура добавлена к объекту и соответствующие имена появляются в Инспекторе объектов.

Добавление новых объектов

Система Delphi, это, прежде всего, среда разработки, базирующаяся на использовании визуальных компонент. Поэтому вы можете добавлять совершенно новые компоненты в палитру компонент. Вы можете создавать компоненты внутри Delphi или вводить компоненты, созданные, как управляющие элементы VBX или OLE 2.0 или же можете использовать компоненты, написанные на C или C++ в виде dll.

Последовательность введения новой компоненты состоит из трех шагов:

- Наследование из уже существующего типа компонент.
- Определение новых полей, свойств и методов.
- Регистрация новой компоненты.

Все это делается через меню системы Delphi - Install Components (Установка или инсталлирование новых компонент).

ТИПЫ ДАННЫХ В DELPHI

С помощью типов данных (объявляемых в разделе type некоторых блоков программы) программист указывает компилятору, как хранить в программе информацию. При объявлении некоторой переменной необходимо указать и ее тип. Одни типы уже определены в самом языке Object Pascal, другие программисту приходится задавать, определять самому. В ранних языках программирования имелось ограниченное число типов данных, и Pascal оказался одним из первых языков, допускающих определение новых типов в самой программе.

Типы данных, определяемые пользователем, обычно задаются в разделе определения типов (type) головной программы или модуля (unit), однако это можно делать и внутри процедур (procedure) или функции (function). Объявления типов и переменных (задаваемых в разделе var) действуют в пределах того блока, в котором они размещены. Вне этого блока ссылаться на такие типы и переменные нельзя, а внутри они заменяют все внешние типы и переменные с тем же именем. Объявленные типы данных можно применять в любом месте области их видимости.

Объявления типов в Pascal являются для компилятора чем - то вроде схем, которые он должен запомнить на случай, если вдруг встретит в программе ссылки на тот или иной тип. Само по себе объявление типа не вносит в программу никаких изменений.

Что же касается объявления переменных var, то они задают компилятору некоторые действия, связанные с ранее объявленными типами. Тип переменной ограничивает, как ее значения, так и операции, которые можно выполнять с этими значениями.

Определения типов и переменных могут размещаться в нескольких местах программы и выглядят следующим образом:

```
type
type1 = type definition1;
type2 = type definition2;
```

Новые типы данных (type definition) определяются в разделе type, и каждому новому типу присваивается имя (type1, type2 и т.д.). В одном разделе type можно объявить несколько типов.

Самое простое определение нового типа состоит из имени типа, определенного ранее:

```
type3 = type1;
```

Новые переменные объявляются в разделе var. Каждой новой переменной сначала присваивается имя или идентификатор (var1, var2 и т.д.), а затем тип:

```
var
var1: type definition1;
```

В одном разделе var можно объявить несколько переменных и присвоить один и тот же тип:

```
var2, var3: type definition2;
```

Программу будет легче читать, если переменным присвоены уже существующие типы:

```
var4: type1;
var5, var6, var7: type2;
```

Синтаксис языка Object Pascal позволяет одновременно конструировать исключительно сложные типы и определения переменных. Определение типов в разделах type тех или иных блоков программы дает возможность использовать эти типы в разных частях программы. Новые типы определяются из типов следующих категории:

- Простые типы для хранения информации в форме чисел и других "упорядоченных" значении.
- Строковые типы для хранения последовательностей символов.
- Структурные типы для одновременного хранения информации разных типов.
- Указательные типы для косвенного обращения к переменным заданных типов.
- Процедурные типы для обращения к процедурам и функциям, рассматриваемым как переменные.
- Вариантные типы для хранения в одной переменной данных различных типов.

Обычно идентификаторы типов используются только при определении новых типов или объявлении переменных. Есть, однако, несколько встроенных в Delphi стандартных функций, в которых имя типа может использоваться, как часть выполняемого оператора. Например, функция SizeOf(T) возвращает количество байтов, занимаемых

переменной T.

Любой реально существующий тип данных, каким бы сложным он ни казался на первый взгляд, представляет собой простые составляющие, которыми процессор может манипулировать. В Object Pascal простые типы данных разбиты на две группы:

- Порядковые типы - представляющие данные разных объемов, которыми процессор может легко манипулировать.
- Действительные типы - приближенно представляющие действительные математические числа.

Разделение простых типов данных на порядковые и действительные типы несколько условно. Точно так же, все данные можно было бы разделить на числа и не числа. Однако в языке Object Pascal порядковые и действительные данные трактуются по - разному, и такое разделение даже полезно.

Порядковые типы

Из простых типов данных, порядковые данные являются самыми простыми. В этих типах информация представляется в виде отдельных элементов. Связь между этими элементами и их представление в памяти компьютера определяет естественные отношения порядка. (Отсюда и название - порядковые).

В языке Object Pascal определены три группы встроенных порядковых типов и два типа, определяемых пользователем. Первые три группы - это целые, символьные и булевы типы данных. Порядковые типы, задаваемые пользователем - это перечисления и под диапазоны значений.

Первый элемент любого порядкового типа имеет номер 0, второй элемент - номер 1 и т.д. Порядковый номер целого значения равен самому значению. Отношения порядка, определяют общие для всех данных порядковых типов, операции. Некоторые стандартные функции такого вида встроены в Object Pascal и представлены в таблице.

Операция	***Описание***
Low(T)	Минимальное значение порядкового типа T.
High(T)	Максимальное значение порядкового типа T.
Ord(X)	Порядковый номер значения выражения порядкового типа. Для целого выражения - просто его значение. Для остальных порядковых типов Ord возвращает физиче-

	ское представление результата выражения, трактуемое как целое число. Возвращаемое значение всегда принадлежит одному из целых типов.
Pred(X)	Предыдущее по порядку значение. Для целых выражений эквивалентно X - 1.
Succ(X)	Следующее по порядку значение. Для целых выражений эквивалентно X+1.
Dec(V)	Уменьшает значение переменной на 1. Эквивалентно V:= Pred(V).
Inc(V)	Увеличивает значение переменной на 1. Эквивалентно V:= Succ(V).

Для всех порядковых типов языка Object Pascal существует операция задания типа для преобразования целых значений в значения соответствующих порядковых типов. Если T - имя порядкового типа (например, символьного), а X - целое выражение, то T(X) возвращает (равно) значение T с порядковым номером X.

Целые типы

В переменных целых типов информация представляется в виде целых чисел, т.е. чисел, не имеющих дробной части. Определенные в Object Pascal целые типы подразделяются на физические (фундаментальные) и логические (общие). При программировании удобнее использовать логические целые типы, которые задают объем переменных в зависимости от типа микропроцессора и операционной среды таким образом, чтобы достигалась максимальная эффективность. Физические целые типы следует применять лишь в тех случаях, когда в первую очередь важны именно диапазон значений и физический объем переменной. В языке Object Pascal определены следующие целые типы:

- Integer
- Shortint
- Smallint
- Longint
- Byte
- Word
- Cardinal

Обратите внимание, что один из этих целых типов назван именно целым (Integer). Это может иногда приводить к путанице, но мы легко

сможем ее избежать, применяя термин "целый" к группе типов, а Integer - к конкретному типу, определяемому в программе этим ключевым словом.

Переменные физических целых типов имеют разные диапазоны значений в зависимости от того, сколько байтов памяти они занимают (что равно значению, возвращаемому функцией SizeOf для данного типа). Диапазоны значений для всех физических типов перечислены в таблице.

Тип	***Диапазон значений***	***Физический формат***
Shortint	- 128 ÷ 127	8 бит, со знаком
Smallint	- 32 768 ÷ 32 767	16 бит, со знаком
Longint	- 2 147 483 648 ÷ 2 147 483 647	32 бит, со знаком
Byte	0 ÷ 255	8 бит, без знака
Word	0 ÷ 65 535	16 бит, без знака

Диапазоны значений и форматы физических целых типов не зависят от микропроцессора и операционной системы, в которых выполняется программа. Они не меняются (или, по крайней мере, не должны меняться) с изменением реализации или версии Object Pascal.

Диапазоны значений логических целых типов (Integer и Cardinal) определяются совершенно иным образом. Как видно из следующей таблицы, они никак не связаны с диапазонами соответствующих физических типов. Обратите внимание, что в Delphi "по умолчанию" задано 32 - разрядное представление чисел, которому соответствуют типы, приведенные во второй и четвертой строках таблицы.

Тип	***Диапазон значений***	***Физический формат***
Integer	- 32 768 ÷ 32 767	16 бит, со знаком (Smalint)
Integer	- 2 147 483 648 ÷ 2 147 483 647	32 бит, со знаком (Longint)
Cardinal	0 ÷ 65 535	16 бит, без знака (Word)
Cardinal	0 ÷ 2 147 483 647	32 бит, без знака (Longint)

Над целыми данными выполняются все операции, определенные для порядковых типов, но с ними все же удобнее работать, как с числами, а не с "нечисленными" порядковыми типами. Как и "живые" числа, данные целых типов можно складывать (+), вычитать (-) и умножать (*). Однако некоторые операции и функции, применяемые к

данным целых типов, имеют несколько иной смысл.

Операция	***Результат***
Abs(X)	Возвращает абсолютное целое значение X.
X Div Y	Возвращает целую часть от деления X на Y.
X Mod Y	Возвращает остаток от деления X на Y.
Odd(X)	Возвращает булево True (истина), если X - нечетное целое, и False (ложь) - в противном случае.
Sqr(X)	Возвращает целый квадрат X (т.е. X*X).

Будьте внимательны при перенесении численных выражений из одного языка в другой. В Basic, например, функция SQR вычисляет квадратный корень. В C/C++ целое деление обозначается косой чертой (/), а в Delphi, косая черта между двумя целыми даст действительный результат с плавающей запятой.

Символьные типы

Смысл символьных данных (один символ) очевиден, когда они выводятся на экран или принтер. Тем не менее, определение символьного типа может зависеть от того, что подразумевать под словом символ. Обычно символьные типы данных задают схему взаимодействия между участками памяти разного объема и некоторым стандартным методом кодирования / декодирования для обмена символьной информацией. В классическом языке Pascal не задано никакой схемы, и в конкретных реализациях применялось то, что на данном компьютере мог использовать каждый пользователь.

В реализациях языка Pascal для первых микропроцессоров была применена 7 - битовая схема, названная ASCII (American Standard Code for Information Interchange - Американский стандартный код для обмена информацией). Эта схема и поныне широко распространена, но информация хранится, как правило, в 8 - битовых участках памяти, что удваивает число возможных представлений символов. В данной версии Delphi определен набор 8 - битовых символов, известный, как расширенный (extended) ANSI (American National Standards Institute - Американский национальный институт стандартов). Для оконных операционных систем фирмы Microsoft это схема включает ограниченное число предназначенных для вывода международных знаков. В стремлении применить более обширный набор международных знаков весь компьютерный мир переходит к 16 - битовой схеме, именуемой UNICODE, в которой первые 256 знаков совпадают с символами, определенными

в схеме ANSI.

Для совместимости со всеми этими представлениями в Object Pascal определены два физических символьных типа (см. таблицу ниже) и один логический.

AnsiChar	Однобайтовые символы, упорядоченные в соответствии с расширенным набором символов ANSI (8 бит).
WideChar	Символы объемом в слово, упорядоченные в соответствии с международным набором символов UNICODE (16 бит). Первые 256 символов совпадают с символами ANSI.

Логический символьный тип именуется Char. В классическом языке Pascal, Char - единственный символьный тип. В Delphi, Char всегда соответствует физическому типу данных AnsiChar.

Ord(C)	Возвращает целое значение, которым символ C представлен в памяти компьютера.
Chr(X)	Преобразует целую переменную X в переменную типа Char с тем же порядковым номером. В Delphi это эквивалентно заданию типа Char(X).
UpCase	Преобразует любую строчную букву в прописную.

Булевы типы

На ранней стадии обучения, программисты осваивают понятие бита, два состояния которого можно использовать для записи информации о чем - либо, представляющем собой один вариант из двух. Бит может принимать значения 0 или 1 - ДА или НЕТ, ВКЛЮЧЕНО или ВЫКЛЮЧЕНО, ВЕРХ или НИЗ. В Object Pascal информация о чем - либо, что можно представить, как ИСТИНА (True) или ЛОЖЬ (False), хранится в переменных булевых типов. Всего таких типов четыре, и они представлены в таблице.

Тип	***Размер***
Boolean	1 байт
ByteBool	1 байт
WordBool	2 байт (объем Word)
LongBool	4 байт (объем Longint)

По аналогии с целыми и символьными типами, подразделяющимися на физические и логические, естественно предположить, что ByteBool, WordBool и LongBool - физические типы, а Boolean - логический. Но в данном случае это не совсем так - все четыре типа различны. Для Object Pascal предпочтителен тип Boolean, остальные определены для совместимости с другими языками программирования и операционными системами.

Переменным типа Boolean можно присваивать только значения True (Истина) и False (Ложь). Переменные ByteBool, WordBool и LongBool могут принимать и другие порядковые значения, интерпретируемые обычно, как False в случае нуля и True - при любом ненулевом значении.

Перечислимые типы пользователя

Обычно данные перечислимых типов содержат дискретные значения, представляемые не числами, а именами. Тип Boolean - простейший перечислимый тип в Object Pascal. Булевы переменные могут принимать два значения, выражаемые именами True и False, а сам тип определен в Object Pascal так, как будто он объявлен в перечислимом виде следующим образом:

```
type
Boolean = (False, True);
```

С помощью типа Boolean в Object Pascal выполняются сравнения, а большинство перечислимых типов - это просто списки уникальных имен или идентификаторов, зарезервированных с конкретной целью. Например, можно создать тип MyColor (Мой цвет) со значениями myRed, myGreen и myBlue (Мой красный, мой зеленый, мой синий):

```
type
MyColor = (myRed, myGreen, myBlue);
```

В этой строке объявлены четыре новых идентификатора - MyColor, myRed, myGreen и myBlue. Идентификатором MyColor обозначен порядковый тип, следовательно, в синтаксисе Object Pascal можно применять этот идентификатор везде, где разрешены перечислимые типы. Остальные идентификаторы - это значения типа MyColor.

Подобно символьным и булевым типам, перечислимые типы не являются числами, и использовать их наподобие чисел не имеет смыс-

ла. Однако перечислимые типы относятся к порядковым типам, так что значения любого такого типа упорядочены. Идентификаторам в списке присваиваются, в качестве порядковых номеров, последовательные числа. Первому имени присваивается порядковый номер 0, второму 1 и т.д.

Поддиапазонные типы пользователя

Переменные поддиапазонного типа содержат информацию, соответствующую некоторому заданному диапазону значений исходного типа, представляющего любой порядковый тип, кроме, поддиапазонного. Синтаксис определения поддиапазонного типа имеет следующий вид:

```
type
Subrange = low value...high value;
```

или

```
type
SR1 = 1...100;
```

Поддиапазонные переменные сохраняют все особенности исходного типа. Единственное отличие состоит в том, что переменной поддиапазонного типа можно присваивать только значения, входящие в заданный поддиапазон. Контроль за соблюдением этого условия задается командой проверки диапазона (Range Checking).

Необходимость явно определять поддиапазонный тип возникает нечасто, но все программисты неявно применяют эту конструкцию при определении массивов. Именно в форме поддиапазонной конструкции задается схема нумерации элементов массива.

Действительные типы

В переменных действительных типов содержатся числа, состоящие из целой и дробной частей. В Object Pascal определено шесть действительных типов. Все типы могут представлять число 0, однако они различаются пороговым (минимальным положительным) и максимальным значениями, которые могут представлять, а также точностью (количеством значащих цифр) и объемом.

Тип	*Минимальное значение*	*Максимальное значение*	*Количество значащих цифр*	*Объем (байт)*
Real	2.9E - 39	1.7E + 38	11 - 12	6
Single	1.5E - 45	3.4E + 38	7 - 8	4
Double	5.0E - 324	1.7E + 308	15 - 16	8
Extended	3.4E - 4932	1.1E + 4932	19 - 20	10
Comp	1.0	9.2E + 18	19 - 20	8
Currency	0.0001	9.2E + 14	19 - 20	8

Тип Real предназначен для совместимости с ранними версиями Delphi и Borland Pascal. Формат этого типа неудобен для семейства процессоров Intel, поэтому операции с типом Real выполняются несколько медленнее операций над остальными действительными типами.

Целые типы представляют целые числа, т.е. числа, дробная часть которых равна нулю. Разница между двумя неодинаковыми целыми числами не может быть меньше единицы. Именно благодаря этому целые числа применяются для обозначения дискретных величин, независимо от того, имеют ли реальные объекты какое - либо отношение к числам. Действительные типы предназначены для представления чисел, которые могут иметь дробную часть, поэтому они полезны для представления величин, которые могут быть довольно близкими, почти непрерывными.

Несмотря на название "действительные", переменные этих типов отличаются от математических действительных чисел. В Object Pascal действительный тип - это подмножество математических действительных чисел, которые можно представить в формате с плавающей запятой и фиксированным числом цифр.

Для невнимательных программистов ситуация усугубляется тем, что в стандартных форматах IEEE (Institute of Electrical and Electronic Engineers - Институт инженеров - электриков и электронщиков), применяемых в программах Delphi и вообще в большинстве программ для Windows, возможно точное представление только чисел с фиксированным числом бит в дробной части. Удивительно, но такое простое число, как 0,1, записывается в расширенном формате IEEE с некоторой погрешностью, пусть и очень небольшой. Из - за этого представление с плавающей запятой оказывается несколько неудобным для программ, в которых сохраняется и выводится фиксированное число десятичных разрядов численных значений. Это относится и к программам, работающим с "живыми" деньгами.

Для частичного решения этой проблемы в Object Pascal определе-

ны два формата с фиксированной запятой. Тип Comp (Computational - вычислительный) содержит только целые числа в диапазоне от - $2^{63}+1$ до $2^{63}-1$, что примерно соответствует диапазону от - $9{,}2x10^{18}$ до $9{,}2x10^{18}$. При программировании операций с любой валютой, разработчикам обычно приходится искать естественный способ записи денежных сумм, в котором целая часть числа определяет количество тенге или долларов, а дробная - тиинов или центов. Если такие значения записывать в переменные типа Comp, придется представлять их в виде целого числа центов.

Этого можно избежать, если воспользоваться типом Currency. Физически значения Currency записываются в память того же объема, что и Comp, как целые числа, однако компилятор не забывает вовремя разделить значение на 10 000 для его приведения в соответствие с денежным знаком и умножить на 10 000 перед записью в память. Это обеспечивает абсолютную точность в четыре десятичных знака после запятой.

В системе Delphi есть модуль System, содержащий ряд процедур обработки данных действительных типов. Наиболее распространенные из них перечислены в таблице. Много полезных процедур и функций содержится также в модулях SysUtils и Math.

Функция	***Возвращаемое значение***
Abs(X)	Абсолютная величина X.
ArcTan(X)	Арктангенс X.
Cos(X)	Косинус числа X (X выражается в радианах, а не в градусах).
Exp(X)	Экспоненциальная функция от числа X.
Frac(X)	Дробная часть X.
Int(X)	Целая часть X. Несмотря на название, возвращает действительное значение (с плавающей запятой), т.е. просто устанавливает нуль в дробной части.
Ln(X)	Натуральный логарифм от числа X.
Pi	Число Пи (3.1416...).
Round(X)	Ближайшее к X целое значение. Возвращает значение целого типа. Условие "ближайшее к X" не работает, если верхнее и нижнее значения оказываются равноудаленными (например, если дробная часть точно равна 0,5). В этих случаях Delphi перекладывает решение на операционную систему. Обычно процессоры Intel решают эту задачу в соответствии с рекомендацией IEEE -

	округлять в сторону ближайшего четного целого числа. Иногда такой подход называют "банкирским округлением".
Sin(X)	Синус X.
Sqr(X)	Квадрат x, т.е. X*X.
Sqrt(X)	Квадратный корень от X.
Trunc(X)	Целая часть X. В отличие от Int, возвращающей действительное значение, Trunc возвращает целое.

Будьте внимательны при переносе численных выражений из одного языка в другой. В Basic функция SQR вычисляет квадратный корень, а функция Sqr из Delphi - квадрат числа. На этом мы заканчиваем рассмотрение Простых типов данных, и переходит к новому типу - строковые.

Строковые типы

В строковых выражениях Delphi поддерживает три физических формата: короткий (ShortString), длинный (LongString) и широкий (WideString). Их можно комбинировать в операторах присваивания и других выражениях (все необходимые преобразования Delphi выполняет автоматически).

Переменные типов AnsiString и WideString - это динамически распределяемые массивы символов, максимальная длина которых ограничивается только наличием оперативной памяти. Разница между ними состоит в том, что в AnsiString знаки записываются в формате Char, а в WideString - в формате WideChar. Обычно вполне достаточно одного типа AnsiString, однако, при работе с международными наборами символов, такими как UNICODE, удобнее использовать WideString.

Тип ShortString - это, по существу, массив Array[0..255] of Char (массивы будут описаны далее). Первый его элемент (с номером ноль) задает динамическую длину строки, которая может принимать значения от 0 до 255 символов. Символы, составляющие саму строку, занимают места (номера) от 1 до 255. Тип ShortString предназначен, в основном, для обеспечения совместимости с ранними версиями Delphi и Borland Pascal.

Логический строковый тип именуется просто String. Отнесение его к типу AnsiString или ShortString задается командой $H. По умолчанию задается {$H+}, и String совпадает с AnsiString. Если задать команду {$H-}, то String будет совпадать с ShortString и иметь максимальную длину, равную 255 символам.

Для совместимости с другими языками программирования в Delphi поддерживается класс строк с конечным нулем. Зарезервированных слов или идентификаторов для этого класса не существует. Строки с конечным нулем состоят из ненулевых символов и оканчиваются символом с порядковым номером 0 (#0). В отличие от типов AnsiString, ShortString и WideString, строки с нулевым окончанием не имеют указателя длины - конец в этих строках обозначается нулем.

В следующей таблице перечислены некоторые процедуры и функции обработки данных строковых типов.

Функция	***Описание***
Concat(sl, s2, s3)	Возвращает последовательное соединение строк. Эквивалентна оператору sl+s2+s3.
Copy(s, pos, len)	Возвращает подстроку длиной максимум len символов, начинающуюся в позиции pos строки s.
Delete(s, pos, len)	Удаляет максимум len символов из строки s, начиная с позиции pos.
Insert(sourse, target, pos)	Вставляет строку source в строковую переменную target, начиная с позиции pos.
Length(s)	Возвращает динамическую длину строки.
Pos(substring, s)	Возвращает место первого вхождения подстроки substring в строку s.
SetLength(s, newlen)	Задает новую динамическую длину newlen строковой переменной s.
SetString	Задает содержимое и длину строки.
Str(x, s)	Преобразует численное значение x в строковую переменную s
StringOfChars	Возвращает строку с конкретным числом символов.
UniqueString	Делает данную строку уникальной со счетом обращений 1.
Val(s, v, code)	Преобразует строку s в соответствующее численное представление v.

Структурные типы

На элементарном уровне наиболее полезными типами данных являются те, в которых содержится численная и строковая (символьная) информация. Объединив несколько образцов этих элементарных типов, можно создавать более сложные типы данных.

Структурные типы данных предоставляют возможность создавать

новые типы, расширяя определения уже существующих, таким образом, чтобы данные новых типов могли содержать более одного значения. Элементами данных структурных типов можно манипулировать, как поодиночке, так и вместе, и эти элементы сами могут быть структурными. Никаких ограничений на подобное вложение одной структуры в другую не существует.

Ниже перечислены структурные типы, определенные в Delphi:

- Записи.
- Массивы.
- Множества
- Файлы.
- Классы.
- Указатели на классы.

Перечисленные типы сами по себе обычно являются не типами, а структурными методами, дополняющими существующие типы.

Записи

С помощью зарезервированного слова Record (запись) в одном типе можно объединять данные разных типов. Общий синтаксис объявления этого типа выглядит следующим образом:

```
record
fieldname1: fieldtype1;
fieldname2, fieldname3: fieldtype2;

case optional tagfield: required ordinal type of
1: variantname1: varianttype3;
2, 3: variantname2: varianttype4;
end;
```

Данное объявление состоит из фиксированной и вариантной частей. Однако вовсе не обязательно вставлять в одно объявление записи обе эти части. Обычно удобнее работать с каждой из этих частей отдельно.

Фиксированные записи

В фиксированной части записи определяется одно или несколько

независимых полей. Каждому полю обязательно присваивается имя и тип:

```
record
fieldname1: fieldtype1;
fieldname2, fieldname3: fieldtype2;
end;
```

Имея доступ к информации в записи, можно обрабатывать всю запись целиком (все поля одновременно) или только отдельное поле. Для обращения к отдельному полю наберите имя записи, точку и идентификатор поля, например:

```
MyRec.Fieldname1
```

Для доступа ко всей записи просто укажите ее имя.

Вариантные записи

Вариантная часть типа Record дает возможность по - разному трактовать область памяти, совместно занимаемую вариантами поля:

```
record
case optional tagfield: required ordinal type of
1: variantname1: varianttype3;
2, 3: variantname2: varianttype4;
end;
```

Термин вариантный в отношении записей не имеет ничего общего с типом Variant, который мы рассмотрим в следующем разделе данной главы. Вариантные поля, несмотря на свое название, никогда не имеют тип Variant. Объявление этого типа в любом месте вариантной части записи запрещено. Все варианты занимают в памяти одно место. Каждый вариант обозначается некоторой постоянной. При желании можно получать доступ ко всем полям всех вариантов одновременно, однако это может иметь смысл только в наиболее простых случаях, когда точно известно, как именно информация каждого варианта записывается в память компьютера.

Каждый вариант обозначается минимум одной константой. Все константы должны быть порядковыми и совместимыми по типу с меткой поля. Необязательное поле - это идентификатор дополнительного

поля в фиксированной части записи, общий для всех вариантов. Обычно с его помощью определяют, когда и к какому варианту обращаться.

Необязательное поле можно не указывать, однако порядковый тип необходим. При отсутствии необязательного поля программе придется выбирать подходящий вариант, каким - то иным образом.

Данные некоторых типов бессмысленно интерпретировать различным образом, и в Object Pascal на некоторые критические типы наложено соответствующее ограничение. Как следствие, в вариантную часть записи нельзя включать длинные строки и переменные типа Variant, а также структурные переменные, содержащие эти типы.

Массивы

Массивы могут быть одно - или многомерными, как в следующем примере:

```
ArrayName: Array [ordinal type] of type definition;
ArrayName: Array [ordinal type1, ordinal type2] of type
definition;
```

Каждый массив содержит некоторое количество элементов информации одного типа. Для обращения к элементу массива надо указать имя массива и индекс элемента, заключенный в квадратные скобки. Обратите внимание, что число элементов массива в каждом измерении задается порядковым типом (Ordinal type). Для этого можно воспользоваться идентификатором некоторого типа (например, Boolean или AnsiChar), однако на практике обычно явно задается поддиапазон целых чисел.
Количество элементов массива равно произведению количеств элементов во всех измерениях.

Пусть, например, целый массив определен следующим образом:

```
var
MyArray: Array[1..10] of Integer;
```

Тогда обращение к его третьему элементу будет выглядеть, как MyArray[3], и выполняться, как к переменной Integer.

Множества

Зарезервированное слово Set (множество) определяет множество

не более чем из 256 порядковых значений:

```
Set of ordinal type
```

Минимальный и максимальный порядковые номера исходного типа (на основе которого определяется множественный тип) должны быть в пределах между 0 и 255. Переменная множественного типа содержит (или не содержит) любое значение исходного порядкового типа. Каждое значение из заданного диапазона может принадлежать или не принадлежать множеству. Рассмотрим следующий пример:

```
type
CharSet = Set of AnsiChar;
```

- тип множества символов ANSI и

```
var
MySet: CharSet;
```

- переменная типа CharSet. Переменная Myset может содержать все элементы множества или не содержать ни одного. При присвоении значения переменной множественного типа, элементы множества (порядковые значения) указываются в квадратных скобках:

```
MySet:= ['A', 'E', 'Г, 'O', 'U', 'Y'];
```

Пустые квадратные скобки задают пустое множество, не содержащее ни одного элемента.

Файловый тип

Тип File предназначен для доступа к линейной последовательности элементов, которые могут представлять данные любого типа, кроме, содержащих типы File и Class. Объявление файлового типа подобно объявлению массива, только без указания числа элементов:

```
File of type1;
```

- файл определенного типа, содержащий записи фиксированной длины. File - файл без типа или "блочный" файл. TextFile - файл с записями переменной длины, разделенными символами CR и LF ("возврат

каретки" и "новая строка").

Механизм ввода - вывода информации, как никакой другой аспект программирования, зависит от языка и реализации. В большинстве случаев предполагается, что программисту незачем вникать во внутреннюю структуру переменных, управляющих вводом - выводом, и при передаче информации следует полностью полагаться на предназначенные для этого процедуры.

Процедурные типы

Программистам, работавшим с ранней версией Turbo Pascal, приходилось основательно изучать способы размещения процедур и функций в памяти компьютера и обращения к ним. Процедурные типы существенно упрощают эту проблему, позволяя трактовать процедуры и функции, как значения, которые можно присваивать переменным и передавать в качестве параметров. Объявление процедурного типа подобно объявлению заголовка процедуры или функции. Единственная разница состоит в том, что опускается имя, следующее обычно после ключевых слов procedure и function.

Вне типа Class в Object Pascal разрешены только глобальные процедурные переменные. Это означает, что процедурной переменной не может быть присвоена процедура или функция, объявленная внутри другой процедуры или функции.

Кроме того, в Delphi можно применять данные процедурных типов, представляющих собой методы. При выполнении программы можно обращаться к определенным методам определенных переменных - объектов. Использование методов в качестве данных процедурных типов позволяет модифицировать поведение переменной - объекта некоторого класса, не объявляя нового класса с новыми методами.

Вариантные типы

Тип Variant предназначен для представления значений, которые могут динамически изменять свой тип. Если любой другой тип переменной зафиксирован, то в переменные типа Variant можно вносить переменные разных типов. Шире всего тип Variant применяется в случаях, когда фактический тип данных изменяется или неизвестен в момент компиляции.

При рассмотрении типа Record мы познакомились с вариантной частью записи, где в одном фрагменте памяти можно хранить информацию нескольких типов. Такой метод недостаточно нагляден. Много

ли пользы от того, чтобы найти в памяти действительное значение с фиксированной запятой и интерпретировать его, как целое. Тип Variant (не имеющий ничего общего с вариантной частью записи) более полезен в управлении данными разных типов. Переменным типа Variant можно присваивать любые значения любых целых, действительных, строковых и булевых типов. Кроме того, вариантные переменные могут содержать массивы переменной длины и размерности с элементами указанных типов.

Все целые, действительные, строковые, символьные и булевы типы совместимы с типом Variant в отношении операции присваивания. Вариантные переменные можно сочетать в выражениях с целыми, действительными, строковыми, символьными и булевыми, при этом все необходимые преобразования Delphi выполняет автоматически.

В Object Pascal определены два особых значения переменной типа Variant. Значение Unassigned применяется для указания, что вариантной переменной пока не присвоено значение, какого - либо типа. Значение Null указывает на наличие в переменной данных неизвестного типа или потерю данных. Разницу между этими двумя значениями трудно уловить. Значение Unassigned присваивается вариантным переменным автоматически при их создании, независимо от того, локальная это переменная или глобальная, и является ли она частью другой, структурной, переменной, такой как запись или массив. Unassigned означает, что к данной вариантной переменной еще не обращались. Null же означает, что к вариантной переменной обращались, но не ввели в нее никакой информации. Таким образом, Null указывает, что значение вариантной переменной недействительно или отсутствует.

Вариантные переменные предоставляют широкие возможности формирования выражений с переменными разных типов. Однако за это приходится платить большим, по сравнению с жестко задаваемыми типами, расходом памяти. К тому же, на выполнение операций с вариантными переменными требуется больше времени.

Интересна проблема использования вариантной переменной, как массива. Элементы массива должны быть одного типа - на первый взгляд, это вполне естественное условие. Однако элементам массива можно присвоить и тип Variant. Тогда каждый элемент может содержать информацию разных типов. Как правило, вариантные массивы создаются с помощью процедуры VarArrayCreate.

Для передачи двоичной информации между контроллерами и серверами обычно применяются вариантные массивы с элементами VarByte. Вариантные массивы этого типа не могут подвергаться никаким преобразованиям. Нельзя также переформатировать содержащую-

ся в них двоичную информацию. Эффективный доступ к ним осуществляется с помощью процедур VarArrayLock и VarArrayUnlock. Элементы вариантного массива не могут иметь тип VarString. Для создания вариантных массивов со строковыми элементами следует выбрать тип VarOleStr.

В таблице ниже перечислены стандартные процедуры и функции обработки вариантных массивов, определенные в модуле System.

Процедура / функция	***Описание***
VarArrayCreate	Создает вариантный массив с заданными пределами и типом.
VarArrayDimCount	Возвращает число измерений данного вариантного массива.
VarArrayHighBound	Возвращает верхний предел измерения вариантного массива.
VarArrayLock	Фиксирует вариантный массив.
VarArrayLowBound	Возвращает нижний предел измерения вариантного массива.
VarArrayOf	Возвращает вариантный массив с указанными элементами.
VarArrayRedim	Изменяет верхний предел вариантного массива.
VarArrayUnlock	Отменяет фиксацию вариантного массива.
VarAsType	Преобразует вариантную переменную в указанный тип.
VarCast	Преобразует вариантную переменную в указанный тип и записывает значение.
VarClear	Сбрасывает значение вариантной переменной.
VarCopy	Копирует одну вариантную переменную в другую.
VarFromDateTime	Возвращает вариантную переменную, содержащую переменную даты/времени.
VarIsArray	Возвращает True, если вариантная переменная является массивом.
VarIsEmpty	Возвращает True, если вариантная переменная содержит Unassigned.
VarIsNull	Возвращает True, если вариантная переменная содержит Null.

VarToDateTime	Преобразует вариантную переменную в значение даты/времени.
VarType	Преобразует вариантную переменную в указанный тип и записывает значение.

Ниже в таблице перечислены типы значении, которые можно присваивать вариантным переменным, и вариантные типы результата.

Тип выражения	***Вариантный тип***
Целый	varlnteger
Действительный, кроме Currency	varDouble
Currency	varCurrency
Строковый и символьный	varString
Булев	varBoolean

Вариантные переменные в отношении операции присвоения совместимы с элементарными типами данных Object Pascal (Integer, Real, String и Boolean), а все нужные преобразования Delphi выполняет автоматически. При необходимости конкретно указать, что вариантное значение надо интерпретировать, как целое, действительное, строковое или булево, следует задать тип в форме TypeName(V), где TypeName - идентификатор соответствующего типа, V - выражение Variant. Задание типа изменяет только способ считывания значения из вариантной переменной, а не само значение внутри ее. Внутреннее же представление изменяется с помощью процедур VarAsType и VarCast.

Указательные типы

Переменная указательного типа содержит значение, указывающее на переменную обычного типа - адрес этой переменной:

pointer

- указатель без типа.

^type1

- указатель с типом. Если исходный тип (тип переменной, на ко-

торую должен ссылаться указатель) еще не объявлен, его надо объявить в том же разделе объявления типов, что и тип указателя. Только исходный тип указателей может совпадать с собственно типом.

Средство	***Описание***
New	Распределяет новый участок динамической памяти и записывает его адрес в переменную указательного типа.
Оператор @	Направляет переменную - указатель на область памяти, содержащую любую существующую переменную, процедуру или функцию, включая переменные, имеющие идентификаторы.
GetMem	Создает новую динамическую переменную заданного объема и записывает ее адрес в переменную указательного типа.

Информация, содержащаяся в переменной указательного типа - это адрес некоторого участка в машинной памяти. Эти значения задаются во время работы программы и могут меняться от одного запуска к другому. Следующие функции обеспечивают доступ к адресной информации в программе и тестирование переменных - указателей.

Функция	***Описание***
Addr	Возвращает адрес указанного объекта.
Assigned	Проверяет, равно ли значение процедурной функции Nil.
Ptr	Преобразует адрес в указатель.

Зарезервированное слово Nil указывает значение указателя, который ни на что не указывает - такие указатели называют неопределенными. В Object Pascal только при определении указателей можно нарушать правило, по которому все указываемые идентификаторы, в том числе идентификаторы типов, должны быть объявлены выше. Здесь можно указать идентификатор еще необъявленного типа, как в следующем примере:

```
type
PointerType = ^NotYetDefinedType;
```

Однако необъявленный тип необходимо объявить ниже в том же блоке объявления типов. Определенный в Object Pascal тип Pointer -

это указатель без типа. Обратиться к переменной через такой указатель невозможно (к переменной типа Pointer нельзя дописывать символ "^"), однако, можно задать ей другой указательный тип. По значениям переменных, тип Pointer совместим с остальными указательными типами.

ПРОГРАММИРОВАНИЕ В DELPHI

В данной главе дается обзор среды программирования Delphi, обсуждаются главные части рабочей среды и охватываются такие важные вопросы, как требования к системным ресурсам и основные части программы, созданной в Delphi.

Требования к системным ресурсам

Система Delphi это высокопроизводительный инструмент создания пользовательских приложений. Текущая версия Delphi является 32 - разрядным компилятором для создания программ, работающих в среде Windows.

Для запуска и работы Delphi требуется хотя бы 486 компьютер на 66 Мгц с 8 Мб оперативной памяти. Небольшие программы, созданные на Delphi, будут работать практически на любом компьютере. Другими словами, они не требуют большой памяти или скорости процессора, что необходимо для самой среды Delphi.

Структура среды программирования

Внешний вид среды программирования Delphi отличается от многих других, которые могут работать в системе Windows. Например, Borland Pascal for Windows 7.0, Borland C++ 4.0, Word for Windows, Program Manager все это MDI (Multiple Document Interface) приложения и выглядят они иначе, чем Delphi. Приложения MDI определяют особый способ управления несколькими дочерними окнами внутри одного большого основного окна.

Среда Delphi следует другой спецификации, называемой Single Document Interface (SDI) и состоит из нескольких отдельно расположенных окон. Это было сделано из - за того, что система SDI близка к той модели приложений, которая используется в самом Windows 95/98.

Если вы используете SDI приложение типа Delphi, то уже знаете, что перед началом работы лучше минимизировать другие приложения, чтобы их окна не загромождали рабочее пространство экрана. Если нужно переключиться на другое приложение, то просто щелкните мышкой по системной кнопке минимизации Delphi (кнопка с черточкой справа вверху любого окна). Вместе с главным окном будут свернуты и все остальные окна среды программирования, освободив место для работы других программ.

Главные составные части среды программирования

Перечислим теперь все основные составные части системы Delphi:

- Дизайнер Форм (Form Designer).
- Окно Редактора Исходного Текста (Editor Window).
- Палитра Компонент (Component Palette).
- Инспектор Объектов (Object Inspector).
- Справочник (On - Line Help).

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

Программисты на Delphi проводят большинство времени, переключаясь между Дизайнером Форм и Окном Редактора Исходного Текста (которое для краткости называют Редактор). Прежде чем вы начнете с системой, убедитесь, что можете распознать эти два важных элемента. Дизайнер Форм Delphi показан на рис.6, а окно Редактора текста на рис.2.

Дизайнер Форм в Delphi столь интуитивно понятен и прост в использовании (для программистов уже работавших с подобными системами), что создание визуального интерфейса программы превращается в детскую игру. Дизайнер Форм первоначально состоит из одного пустого окна (рис.6), которое вы заполняете всевозможными объектами (элементами или компонентами), выбранными на Палитре Компонент.

Несмотря на всю важность Дизайнера Форм, местом, где программисты обычно проводят основное время, является Редактор. Логика является движущей силой любой программы и Редактор - то место, где вы ее “кодируете”, т.е. пишете саму программу.

Палитра (Панель) Компонент позволяет вам выбрать нужные объекты для размещения их на Дизайнере Форм. Для использования объекта с Палитры Компонент просто щелкните мышкой по одному из объектов, а потом щелкните в определенном месте Дизайнера Форм. Выбранный вами объект появится на проектируемом окне и им можно манипулировать с помощью мыши.

Палитра Компонент использует постраничную (страницы или вкладки - закладки) группировку объектов. Вверху Палитры находится набор закладок - Standard, Additional, Dialogs и т.д. Если вы щелкнете мышью по одной из закладок, то сможете перейти на следующую

страницу (вкладку) Палитры Компонент. Принцип разбиения на страницы широко используется в среде программирования Delphi и его легко можно использовать в своей программе. (На странице Additional есть компоненты для организации страниц с закладками сверху и снизу).

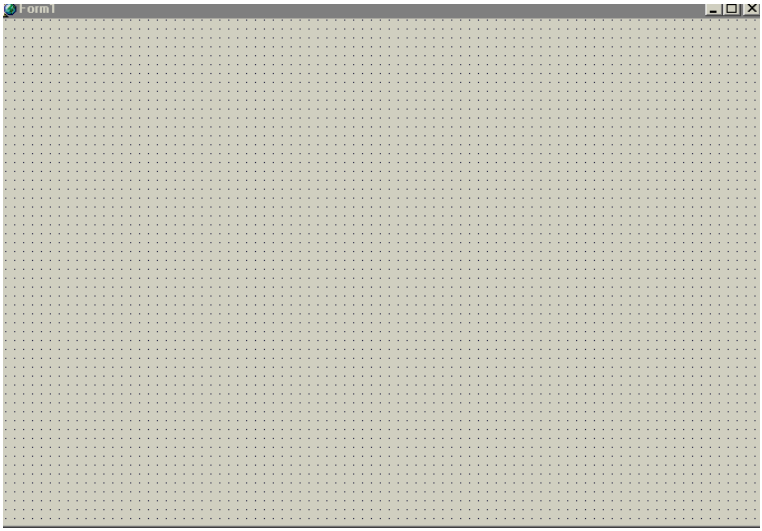

Рис.6. Окно Формы.

Предположим, вы поместили компонент TEdit (Окно редактирования) на Форму - теперь вы можете переместить это окно на другое место, просто подцепив его мышкой. Вы также можете использовать границу, прорисованную вокруг объекта для изменения его размеров. Большинством других компонент можно манипулировать таким же образом. Однако, невидимые во время выполнения программы компоненты, типа TMenu или TDataBase своей формы не меняют.

Слева от Дизайнера Форм вы можете видеть Инспектор Объектов (рис.3). Заметим, что информация в Инспекторе Объектов меняется в зависимости от объекта, выбранного на Форме (для выбора объекта нужно щелкнуть по нему мышкой). Важно понять, что каждый компонент является настоящим объектом, и вы можете менять его вид и поведение с помощью Инспектора Объектов.

Инспектор Объектов состоит из двух страниц, каждую из которых можно использовать для определения поведения выделенного компо-

нента. Первая страница это список Свойств (Properties), вторая - список Событий (Events). Если нужно изменить что - нибудь, связанное с определенным компонентом, то обычно это делается именно в Инспекторе Объектов. Например, вы можете изменить имя, положение и размер компонента TLabel (Ярлык) изменяя свойства Caption, Left, Top, Height, и Width.

Вы можете использовать закладки вверху Инспектора Объектов для переключения между страницами Свойств и Событий. Страница Событий непосредственно связана с Редактором и если вы дважды щелкнете мышкой по правой стороне какого - нибудь пункта, то соответствующий данному событию код автоматически запишется в Редактор. Сам Редактор немедленно получит фокус (выйдет на передний план экрана) и вы имеете возможность добавить код обработчика данного события.

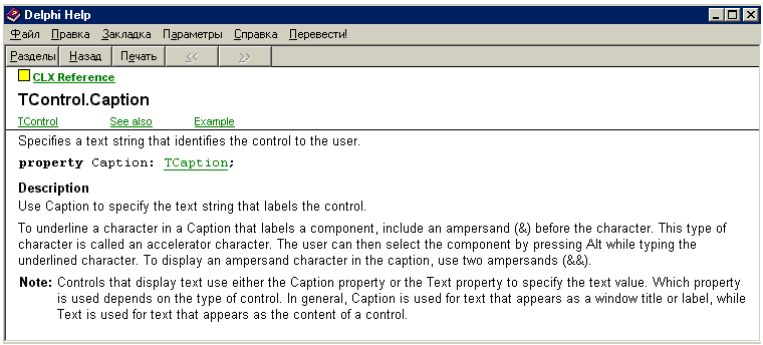

Рис.7. Справочник.

Последняя, важная часть среды Delphi это Справочник (On - Line Help). Для доступа к этому инструменту нужно просто выбрать в системном (Главном) меню пункт Help, а затем команду Contents. На экране компьютера появится Справочник, показанный на рис.7.

Справочник является контекстно - зависимым, поэтому при нажатии клавиши F1, вы получите подсказку, соответствующую текущей ситуации. Например, находясь в Инспекторе Объектов, выберите (щелкните мышкой) какое - нибудь свойство и нажмите F1 - вы получите справку о назначении данного свойства. Если в любой момент работы в среде Delphi возникает неясность или затруднение - нажмите F1 и необходимая информация сразу появится на экране.

Дополнительные элементы

В данном параграфе мы рассмотрим три инструмента, которые можно воспринимать, как вспомогательные возможности среды программирования Delphi:

- Главное или Системное меню (System Menu).
- Панель управления с кнопками для быстрого доступа к различным возможностям системы (SpeedBar или ToolBar).
- Редактор картинок (Image Editor).

Главное меню предоставляет быстрый и гибкий способ доступа к различным возможностям среды Delphi, потому что может управляться набором “горячих клавиш”, которые позволяют управлять работой всей системы непосредственно с клавиатуры. Это удобно еще и потому, что здесь используются слова или короткие фразы, более точные и понятные, нежели иконки или пиктограммы.

Вы можете использовать меню для выполнения широкого круга задач и, скорее всего, для наиболее общих задач, вроде открытия и закрытия файлов, управления отладчиком или настройкой всей среды программирования. Панель SpeedBar находится непосредственно под меню, слева от Палитры Компонент (рис.8). Панель SpeedBar выполняет многое из того, что можно сделать и через Главное меню. Если задержать мышь над любой из иконок на SpeedBar, то вы увидите, что появится подсказка, объясняющая назначение данной кнопки. Первые три кнопки этой панели совпадают со стандартными кнопками Windows - Создать, Открыть и Сохранить.

Рис.8. Панель SpeedBar.

Редактор Картинок, показанный на рис.9, работает аналогично программе графического редактора Paint из Windows. Вы можете получить доступ к этому модулю, выбрав пункт Главного меню Tools (Инструменты), а затем команду Image Editor (Редактор картинок).

А теперь перейдем к рассмотрению тех элементов системы, которые любой программист на Delphi постоянно использует в повседневной жизни для разработки любых прикладных программ.

Стандартные компоненты

Для дальнейшего знакомства со средой программирования Delphi,

потребуется рассказать о первой странице Палитры Компонент - Панели Стандартная (Более подробно панели инструментов будут рассмотрены в главе Панели инструментов).

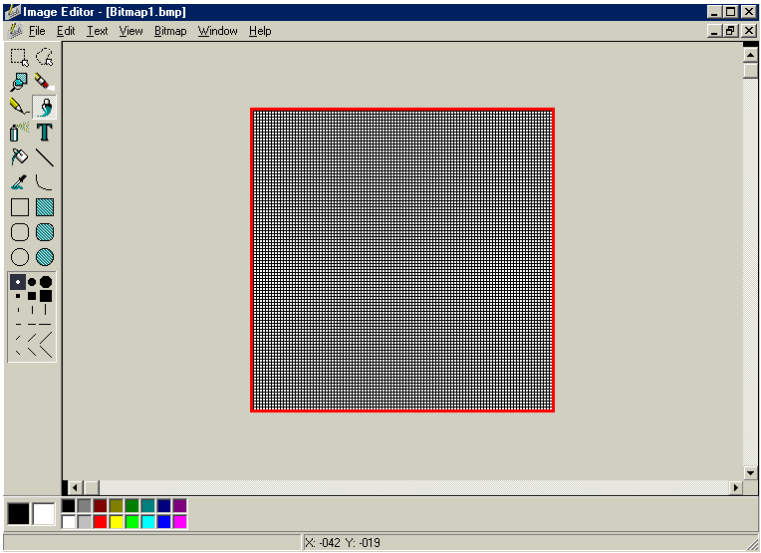

Рис.9. Редактор Картинок.

На первой странице Палитры Компонент размещены 17 основных объектов (рис.10), необходимых для разработки практически любой программы (Самая левая кнопка Панели просто включает указатель и определенным объектом не является). Практически невозможно обойтись без кнопок, списков, окон ввода и т.д. Все эти объекты такая же часть системы Windows, как ярлык или папка.

Набор и порядок компонент на каждой странице являются конфигурируемыми. Так, вы можете добавить к имеющимся компонентам новые, изменить их количество и порядок.

Рис.10. Стандартная Палитра Компонент.

Если вам нужна дополнительная информация о некотором объек-

те, то выберите (один раз щелкните мышкой) объект на любой Палитре и нажмите клавишу F1 - появится Справочник с полным описанием данного компонента.

Инспектор Объектов

Ранее мы вкратце рассматривали Инспектор Объектов (Object Inspector), а теперь нужно исследовать этот важный инструмент системы Delphi более подробно. Инспектора Объектов используется для изменения характеристик любого выделенного объекта, помещенного на Форму, также как для изменения свойств самой Формы.

Лучший способ для изучения Инспектора объектов это поработать с ним. Для начала откройте новый проект, выбрав пункт меню File (Файл), New Project (Новый проект). Затем поместите на Форму объекты TMemo, TButton и TListBox, как показано на рис.11.

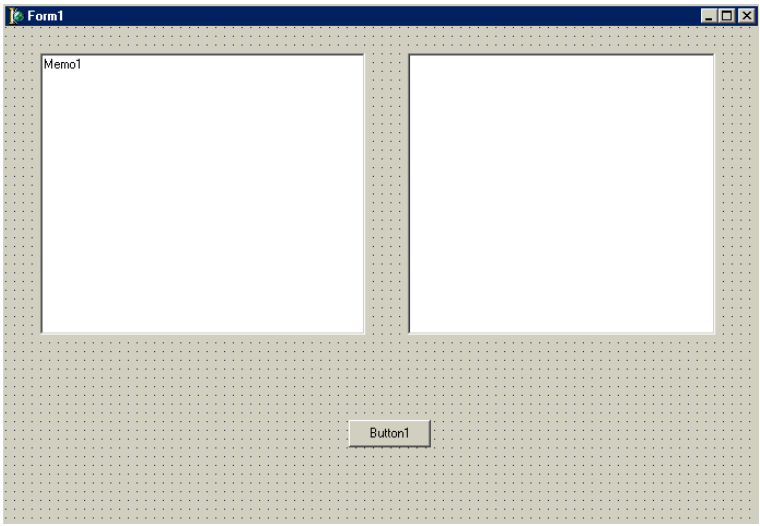

Рис.11. Простой объект TForm с компонентами TMemo, TButton и TListBox.

Вначале рассмотрим работу со Свойствами объекта на примере свойства Ctl3D ("по умолчанию" включено). Выберите Форму, щелкнув по ней мышкой, перейдите в Инспектор Объектов и несколько раз с помощью двойных щелчков мышью переключите значение свойства

Ctl3D. Обратите внимание, что это действие радикально меняет внешний вид Формы. А изменение свойства Ctl3D Формы автоматически изменяет свойство Ctl3D каждого дочернего окна, помещенного на эту Форму.

Поставьте теперь значение свойства Ctl3D Формы в положение True (Истина). Нажмите далее клавишу Shift и щелкните по объекту TMemo, а затем по окну TListBox. Теперь оба объекта имеют по краям маленькие квадратики, показывающие, что эти объекты выбраны (выделены).

Выбрав, таким образом, два или более объектов одновременно, вы можете выполнить над ними большое число операций. Например, можно передвигать выделенные объекты по Форме. Затем попробуйте выбрать пункт Главного меню Edit (Редактировать), щелкнуть команду Size (Размер) и в появившемся окне установить оба поля Ширину (Width) и Высоту (Height) в положение Grow to Largest, как показано на рис.12.

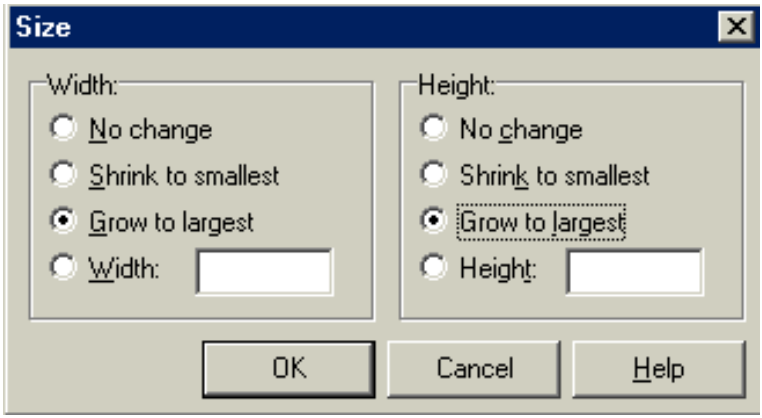

Рис.12. Изменение размеров.

Теперь оба выделенных объекта стали одинакового размера. Выберите затем в пункте Главного меню Edit, команду Align (Выравнивание) и установите выравнивание по горизонтали в положение Center - По центру (рис.13).

Поскольку были выбраны два компонента (объекта на Форме), то содержимое Инспектора Объектов изменится - теперь он будет показывать только те поля, которые являются общими для выделенных ранее объектов. Это означает, что изменения в свойствах, произведенные

вами повлияют не на один, а на все выбранные объекты.

Рассмотрим теперь изменение свойств объектов на примере свойства Color (Цвет). Есть три способа изменить его значение в Инспекторе Объектов:

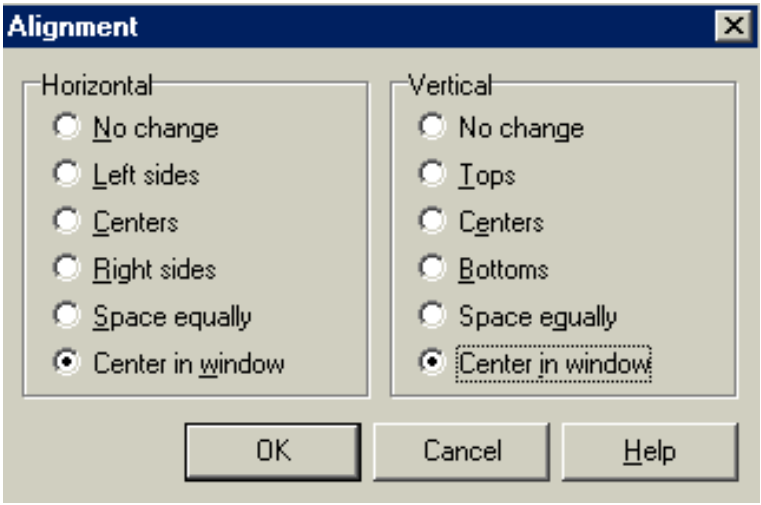

Рис.13. Окно выравнивания.

- Первый - просто напечатать (набрать на клавиатуре) имя цвета (например, clRed) или номер цвета в поле ввода свойства Color окна Инспектора Объектов.
- Второй путь - нажать на маленькую стрелку справа от этого поля и выбрать цвет из открывшегося списка.
- Третий путь - дважды щелкнуть на поле ввода свойства Color. При этом появится диалог (диалоговое окно) выбора цвета.

Свойство объекта Font (Шрифт) работает подобно свойству Color. Чтобы посмотреть эту работу, выберите свойство Font для объекта TMemo и дважды щелкните мышкой на его поле ввода. Появится диалог настройки шрифта, который показан на рис.14. Выберите, например, шрифт New Times Roman и установите какой - нибудь очень большой размер, например, 72. Затем измените цвет (Color) шрифта с помощью меню типа ComboBox в нижнем левом углу окна диалога. После нажатия кнопки OK окна диалога, вы увидите, что вид текста в объекте TMemo радикально изменился.

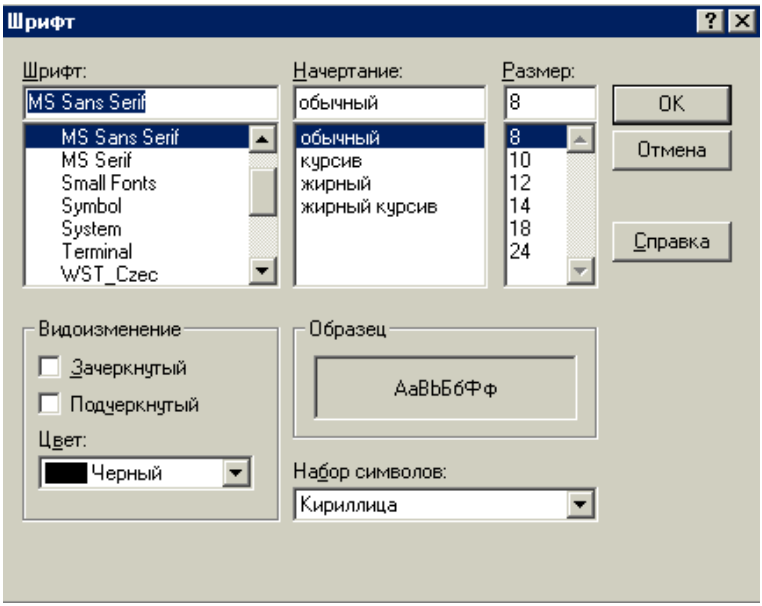

Рис.14. Диалог выбора шрифта.

В завершение этого обзора по Инспектору Объектов дважды щелкните на свойстве Items объекта ListBox. Появится диалог, в котором вы можете ввести строки для отображения в меню ListBox. Напечатайте (наберите на клавиатуре) несколько слов, по одному на каждой строке и нажмите кнопку OK. Набранный текст, в виде пунктов меню отобразится в окне объекта ListBox.

TButton - исходный текст и заголовки

Рассмотрим теперь еще несколько возможностей Инспектора Объектов и Дизайнера Форм. Создайте новый проект и поместите на Форму объект TMemo, а затем TEdit так, чтобы он наполовину перекрывал TMemo.

Выделите объект TEdit и выберите пункт Главного меню Edit, команду Send to Back (Поместить позади), что приведет к перемещению объекта TEdit вглубь Формы, за объект TMemo. Такая процедура называется изменением Z - порядка компонент (объектов). Буква Z используется потому, что обычно математики обозначают третье измерение

буквой Z. Так, X и Y используются для обозначения ширины и высоты, а Z используется для обозначения глубины.

Если вы “потеряли” на Форме какой - то объект, то найти его можно в списке ComboBox, который находится в верхней части окна Инспектора Объектов. Поместите кнопку TButton в нижнюю часть Формы, показанной на рис.15. Теперь растяните окно Инспектор Объектов так, чтобы свойства Name (Имя) и Caption (Заголовок) были одновременно видны на экране.

Теперь измените Имя кнопки (обозначенной, как Button1) на слово Terminate. Обратите внимание, что и Заголовок (Caption) в тот же момент поменялся и примет значение Terminate. Такое двойное изменение наблюдается только в том случае, если ранее свойство Caption не изменялось. Текст, который вы видите на поверхности кнопки это содержимое свойства Caption (ее Заголовок), а свойство Name (Имя объекта) служит исключительно для внутренних ссылок в программе. Если вы откроете сейчас окно Редактора (дважды щелкнув по Форме), то увидите следующий фрагмент кода:

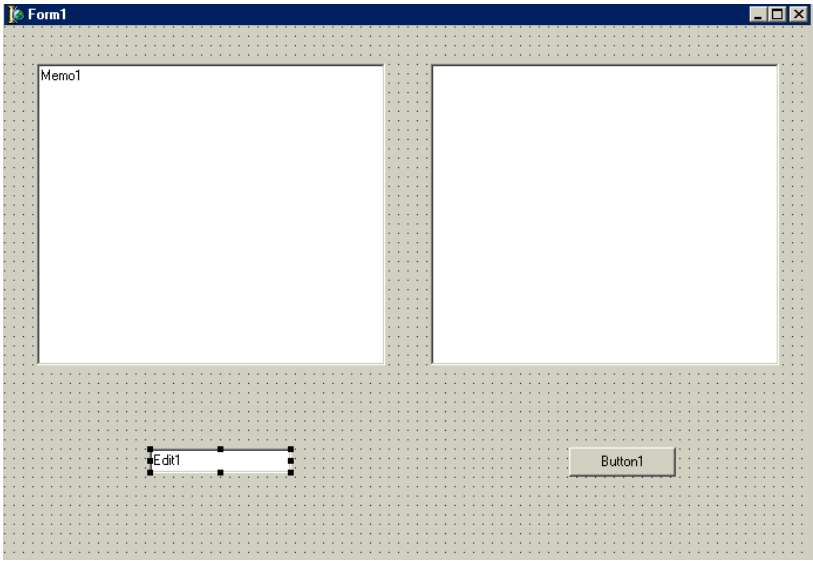

Рис. 15. Объект TEdit.

```
TForm1 = class(TForm)
Edit1: TEdit;
```

```
Memo1: TMemo;
Terminate: TButton;

private
{ Private declarations }

public
{ Public declarations }
end;
```

В этом фрагменте кода программы кнопка TButton называется Terminate из - за того, что вы присвоили это название свойству Name. Обратите внимание, что TMemo и TEdit имеют имена, которые присваиваются им "по умолчанию".

Вернитесь на Форму (щелкнув в любом ее месте) и дважды щелкните мышкой объект TButton. Вы сразу попадете в окно Редактора, в котором увидите фрагмент кода кнопки:

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

Данный код был создан автоматически, и будет выполняться всякий раз, когда во время работы программы пользователь нажмет кнопку Terminate. Вдобавок, вы можете видеть, что определение класса (Class) в начале файла включает теперь ссылку на метод TerminateClick (Щелчок по кнопе Terminate).

Теперь самое время написать строчку кода для выполнения некоторых действий при нажатии на эту кнопку. Это очень простой код, состоящий всего из одного слова Close (Закрыть):

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

Когда этот код исполняется, главная Форма (а значит и все приложение) закрывается и исчезает с экрана. Для проверки кода запустите программу (кнопка с треугольником на Панели управления проектом) и нажмите кнопку Terminate. Если все сделано правильно, программа закроется, и вы вернетесь в режим дизайна Форм.

Перейдите теперь в Инспектор Объектов и измените значение свойства Name для этой кнопки на любое другое, например, "OK". Нажмите клавишу Enter для внесения изменений и посмотрите в Редактор - вы увидите, что код кнопки, написанный вами изменится:

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

Заметьте, что аналогичные изменения произошли и в определении класса:

```
TForm1 = class(TForm)
Edit1: TEdit;
Memo1: TMemo;
Ok: TButton;

procedure OkClick(Sender: TObject);

private
{ Private declarations }

public
{ Public declarations }
end;
```

Сохранение программы

Вы приложили определенные усилия по созданию некоторой программы и можете захотеть ее сохранить на жестком диске (винчестере) компьютера. Это позволит загрузить написанную программу в любое время и снова с ней поработать - дополнить, изменить или преобразовать.

Первый шаг - это создать поддиректорию для хранения программы. Лучше всего создать директорию, где будут храниться все ваши программы, а в ней создать поддиректорию для данной конкретной программы. Например, вы можете создать директорию PROJECT и внутри нее вторую директорию PROBA (поддиректорию), которая будет содержать программу, написанную вами выше.

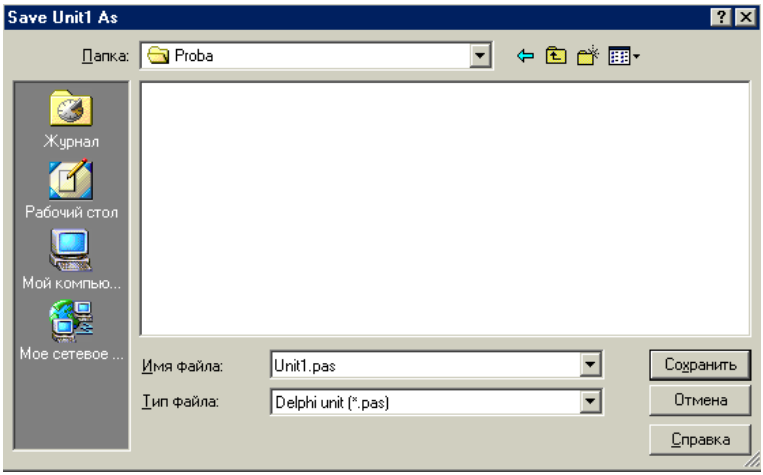

Рис.16. Окно сохранения проекта.

После создания поддиректории для хранения вашей программы, нужно выбрать пункт Главного меню File (Файл), команду Save Project (Сохранить проект) и сохраните два файла. Первый это модуль (unit), над которым вы работали, второй - главный файл проекта, который управляет всей вашей программой (рис.16).

Сохраните модуль под именем MAIN.PAS, а сам проект под именем TIP.DPR. Именно такой способ является стандартным для сохранения всех создаваемых проектов в системе Delphi.

ПРОЕКТ DELPHI

Любой проект в Delphi состоит, по - крайней мере, из шести файлов, которые связаны с ним:

1. Главный файл проекта, изначально ("по умолчанию") называется PROJECT1.DPR.
2. Первый и основной модуль программы (unit), который автоматически создается в начале работы. "По умолчанию" файл называется UNIT1.PAS, но его можно назвать любым другим именем, например, MAIN.PAS.
3. Файл главной, основной Формы, который "по умолчанию" называется UNIT1.DFM и используется для сохранения информации о внешнем виде главной Формы.
4. Кроме того, автоматически создается файл PROJECT1.RES, который содержит иконку (картинку) проекта.
5. Далее создается файл, который "по умолчанию" называется PROJECT1.OPT и является текстовым файлом для сохранения установок, связанных с данным проектом. Именно здесь сохраняются установленные вами директивы или параметры компилятора.
6. И, наконец, создается файл PROJECT1.DSK, который содержит информацию о состоянии рабочего пространства всей среды.

Разумеется, если сохранить проект под другим именем, то изменят название и файлы с расширением RES, OPT и DSK. После компиляции программы получаются файлы с расширениями:

- DCU - скомпилированные модули, которые содержит программа.
- EXE - исполняемый файл самой программы.
- DSM - служебный файл для запуска программы в среде Delphi. Этот файл очень большой и рекомендуется стирать его по окончанию работы программы.

Перейдем теперь к рассмотрению основных пунктов Главного меню (рис.17) программы и кратко опишем их функциональные возможности.

Рис.17. Главное меню.

Пункт меню File

Если нужно сохранить разработанный проект, то следует выбрать пункт Главного меню File (щелкнув по нему мышкой или с помощью комбинации клавиш Alt, F - клавиши нажимаются последовательно, одна за другой). Откроется ниспадающее меню, показанное на рис.18 и можно видеть, что здесь имеется пять секций:

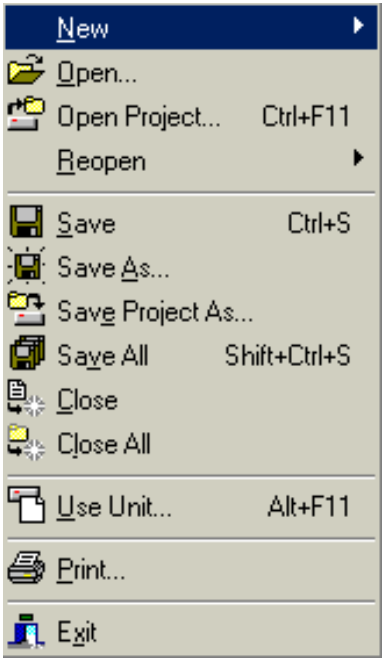

Рис.18. Ниспадающее меню File.

- Первая (верхняя) секция дает возможность создавать и открывать проект.
- Вторая позволяет сохранять и закрывать проект.
- Третья выводит список использованных модулей.
- Четвертая управляет печатью содержимого файлов проекта.
- Пятая позволяет выйти из системы Delphi

Большинство операций из пункта меню File можно выполнить с помощью Менеджера Проекта (Project Manager), который вызывается из пункта Главного меню View (Просмотр, Обзор). Некоторые операции доступны и через SpeedBar - Панель инструментов системы. Данная стратегия типична для Delphi - она предоставляет несколько путей для решения одной и той же задачи, и вы сами можете решать, какой из них более эффективен в данной ситуации (эти возможности полностью аналогичны возможностям редактора Word).

Каждая строка пункта меню File подробно объяснена в Справочнике (Help). Выберите меню File (нажав, например, Alt, F) и нажмите F1 - на экране появится экран справочника, как показано на рис.7. Большинство из пунктов Главного меню File системы Delphi очевидны:

- New (открывается в дополнительное меню) - начинает новый

проект (Application), новую Форму (Form), новый модуль (Unit) или другие элементы программы, как показано ниже на рис.19.

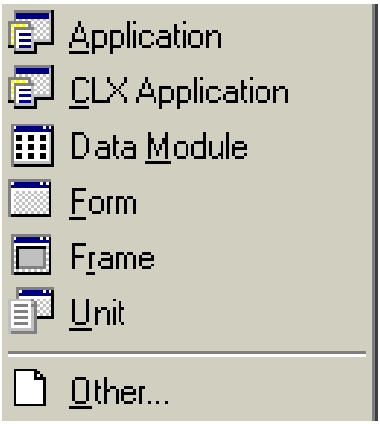

Рис.19. Дополнительное меню раздела New.

- Open Project - открывает существующий проект.
- Reopen - позволяет заново открывать ранее созданные проекты.
- Open (Открыть) - открывает при необходимости любой модуль или просто текстовой файл. Если модуль описывает Форму, то эта Форма тоже появится на экране. При создании нового модуля система Delphi дает ему имя "по умолчанию". Вы можете изменить это имя на что - нибудь более осмысленное (например, MAIN.PAS) с помощью пункта Save As (Сохранить как) меню File.
- Save (Сохранить) - сохраняет только редактируемый файл, но не весь проект.
- Close (Закрыть) - удаляет открытый файл из окна Редактора.

Нужно обратить внимание, что вы должны регулярно сохранять свой проект через меню File, командой Save All (Сохранить все) либо нажатием на клавиатуре комбинации клавиш Shift + Ctrl + S.

Управление проектом

Теперь, когда вы знаете о создании проекта с помощью пункта меню File, перейдем к Менеджеру Проектов, который помогает управлять проектом. Менеджер Проектов, окно которого показано на рис.20, разделен на две части. Верхняя - панель с управляющими кнопками. Нижняя - список модулей, входящих в проект.

Вы можете использовать кнопку с крестиком для удаления файлов из проекта, а кнопка New позволяет создавать новые объекты включаемые в проект. Через пункт Главного меню Project, Add to Project можно добавлять уже существующие файлы или модули в проект. Эти изменения влияют на файлы с исходным текстом, то есть, если добавить в проект модуль, то ссылка на него появится в файле с расшире-

нием DPR.

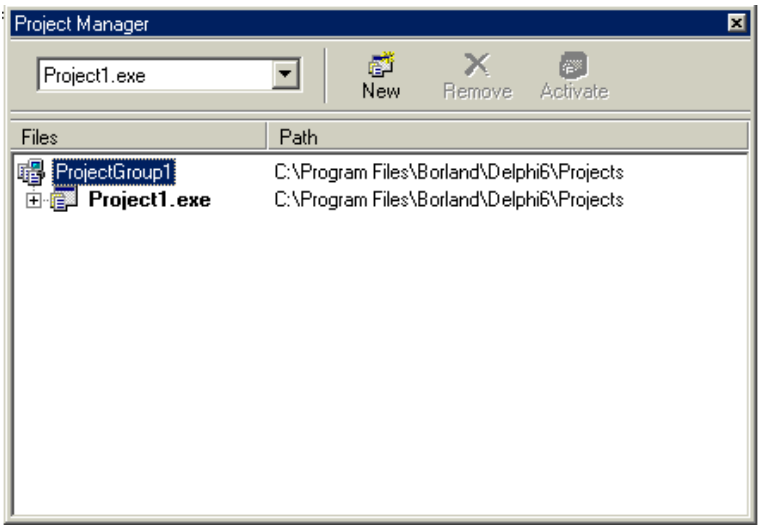

Рис.20. Окно Менеджера проектов.

Обзор других пунктов меню

Далее рассмотрим, но менее подробно другие основные пункты Главного меню системы, а именно - Edit (Редактирование), Search (Поиск), View (Просмотр), Project (Проект) и Run (Счет, Выполнение).

Пункт меню Edit

Пункт Главного меню Edit (Редактирование), показанный на рис.21 содержит команды Undo (Отменить) и Redo (Вернуть), которые могут быть очень полезны при работе в Редакторе, например, для устранения последствий при неправильных действиях. Если вы случайно удалили нужный фрагмент текста, можно использовать команду Undo (Эти команды работают точно так же, как в редакторе Word).

Отметим, что Справочник (On - Line Help) объясняет, как нужно использовать пункт Главного меню Tools и команду Environment для настройки команды Undo. Эта команда дает возможность ограничить допустимое количество команд Undo, и может очень пригодиться, если вы работаете на машине с ограниченными ресурсами.

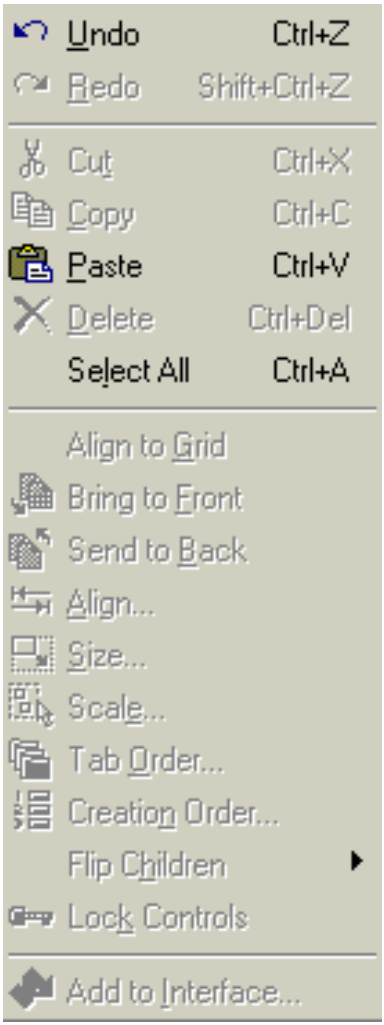

Рис.21. Пункт меню Редактирование (Edit).

Команды Cut (Вырезать), Copy (Копировать), Paste (Вставить) и Delete (Удалить) работают так же, как во всех остальных приложениях Windows (например, в редакторе Word), но их можно применять не только к тексту, но и к визуальным компонентам, объектам Delphi.

Команды Bring To Front, Send To Back, Align и Size обсуждались нами ранее. Оставшиеся пункты меню помогают быстро оформить внешний вид создаваемой Формы.

Пункт меню Search

В пункте Главного меню Search (Поиск), показанном на рис.22 есть команда Find Error (Поиск ошибки), которая поможет отследить ошибку при выполнении вашей программы. Когда в сообщении об ошибке указан ее адрес, вы можете выбрать пункт меню Search, команду Find Error и ввести этот адрес. Если это окажется возможным, то среда Delphi переместит вас именно в то место программы, где произошла ошибка. Остальные пункты меню вполне очевидны и используются так же, как в любых других приложениях системы Windows.

Пункт меню View

Пункт Главного меню View (Просмотр), вид которого приведен на рис.23 состоит из следующих команд:

- Project Manager - вызов, включение Менеджера Проекта.

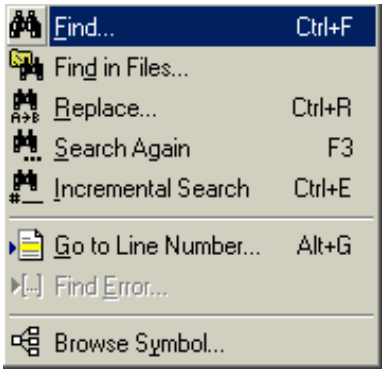

Рис.22. Пункт меню Поиск (Search).

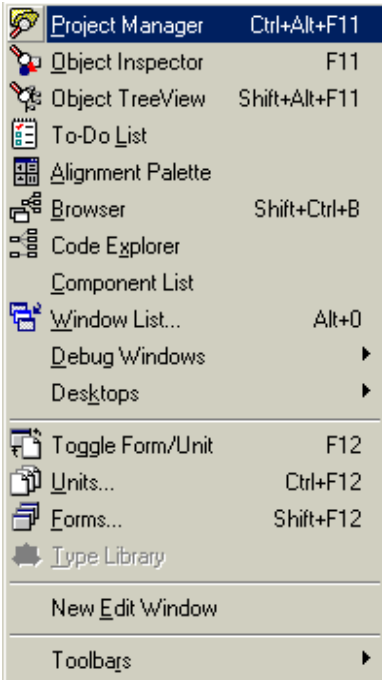

Рис.23. Пункт меню Просмотр (View).

- Установка, показывать или нет Object Inspector (Инспектор Объектов) на экране компьютера.
- Установка, показывать или нет Alignment Palette (Палитра выравнивания). То же самое доступно из пункта меню Edit, командой Align.
- Browser - вызов средства для просмотра иерархии объектов программы, поиска идентификаторов в исходных текстах и т.п.
- Watch, Breakpoint и Call Stack - связаны с процедурой отладки программы.
- Component List - список компонент, альтернатива Палитре Компонент. Используется для поиска компонента по имени или при отсутствии мыши.
- Window List - список окон, открытых в среде Delphi.
- Toggle Form/Unit - переключение между Формой и соответствующим модулем, выбор модуля или Формы из списка.
- New Edit Window - открывает дополнительное окно Редактора, которое бывает полезно, если нужно, например, просмотреть две разных версии одного файла.
- ToolBars (Панель инструментов) - отображает на экране компьютера дополнительное меню для включения различных панелей инструментов.

Пункт меню Project

В пункте Главного меню Project (Проект), показанном на

рис.24, в проект можно добавить (Add to Project) или убрать (Remove from Project) файлы из проекта. Добавить новый (Add New Project) или уже существующий (Add Existing Project) проект. Можно откомпилировать (Compile Project) или Перестроить (Build Project) проект. Если выбрать Compile, то Delphi перекомпилирует только те модули, которые изменялись со времени последней компиляции.

Команда Build all Project (Перестроить все проекты) перестроит, а команда Compile all Project перекомпилирует все модули, исходные тексты которых доступны в данном проекте. Команда Syntax Check (Проверка синтаксиса) только проверяет правильность кода программы, но не обновляет DCU файлы.

В самом низу меню Project имеется команда Options (Параметры), которая позволяет устанавливать параметры всего проекта в целом.

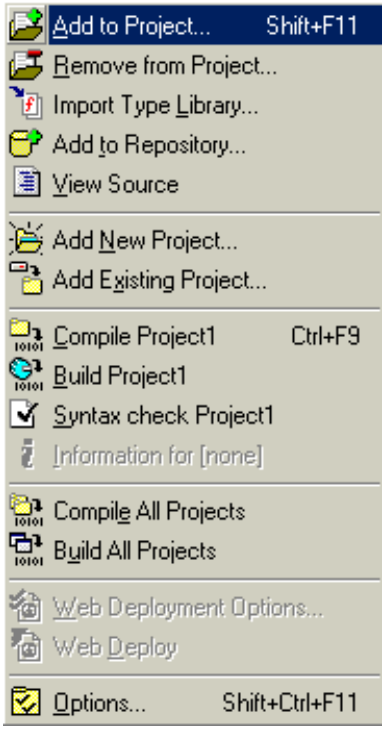

Рис.24. Пункт меню Проект (Project).

Пункт меню Run

Можно использовать пункт Главного меню Run (Запуск), вид которого приведен на рис.25 для компиляции и запуска программы на работу (выполнение) и для указания параметров командной строки для передачи их в программу. Здесь же имеются опции - параметры для включения режима отладки.

Пункт Options из меню Project

Пункт меню Options (Параметры) наиболее сложная часть меню Project. Это центр управления, из которого вы можете изменять установки проекта и всей рабочей среды системы Delphi.

Диалог из пункта Options, меню Project содержит несколько страниц (вкладок или закладок):

1. На странице Forms (Формы) перечислены все Формы, вклю-

ченные в проект. Здесь вы можете указать, нужно ли автоматически создавать Форму при старте программы или вы ее создадите сами после запуска системы Delphi.

2. На странице Application (Приложение) вы определяете элементы программы такие, как заголовок, файл помощи и иконка.

3. Страница Compiler (Компилятор) включает установки для генерации кода, управления обработкой ошибок времени выполнения, синтаксиса, отладки и т.д.

4. На странице Linker (Связь) можно определить условия для процесса линковки приложения

5. Страница Directories / Conditionals (Директории / Условия) позволяет указывать директории, специфичные для данного проекта.

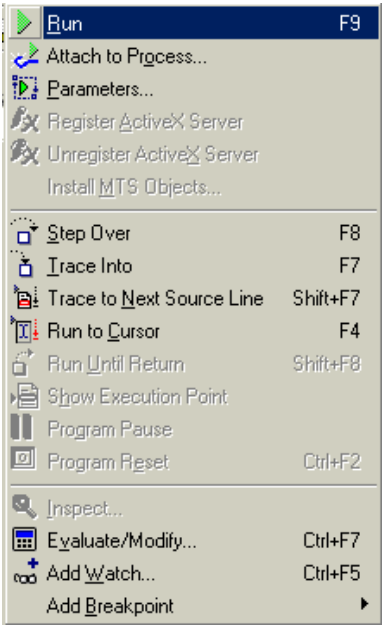

Рис.25. Пункт меню Счет (Run).

Все перечисленные выше установки параметров проекта сохраняются в текстовом файле с расширением OPT, и вы всегда можете вручную их исправить. Опишем теперь каждую из перечисленных выше страниц (вкладок или закладок) более подробно.

Страница Forms

На странице Forms, которая показана на рис.26, можно выбрать главную Форму проекта. Изменения, которые вы сделаете, отобразятся в соответствующем DPR файле. Например, в следующем проекте, Form1 является главной, поскольку появляется первой в главном блоке программы:

```
program Project1;
Uses Forms, Unit1 in 'UNIT1.PAS'
  {Form1}, Unit2 in 'UNIT2.PAS' {Form2};

  {$R *.RES}

  begin
```

```
Application.CreateForm(TForm1, Form1);
Application.CreateForm(TForm2, Form2);
Application.Run;
end.
```

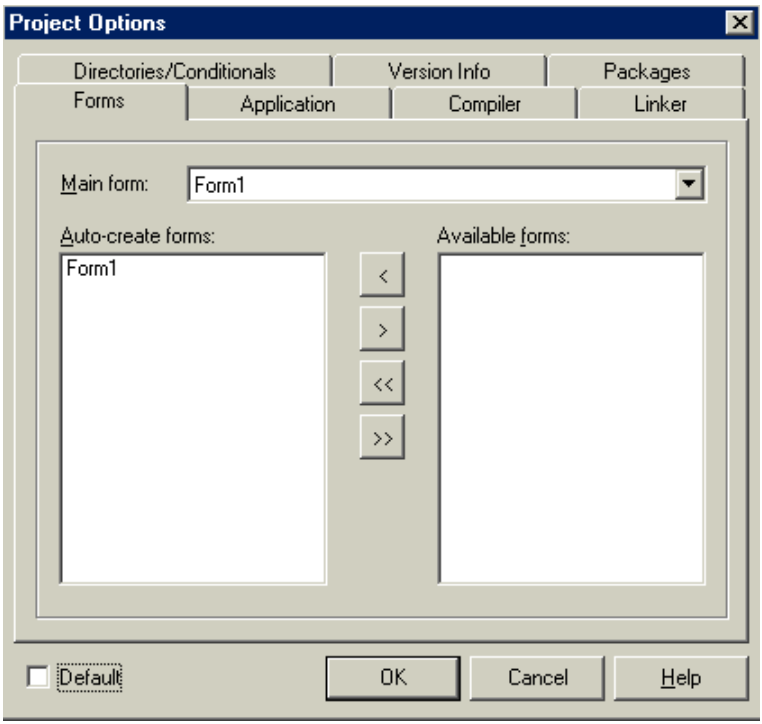

Рис.26. Страница Форм.

Если изменить код программы так, чтобы он имел вид

```
begin
Application.CreateForm(TForm2, Form2);
Application.CreateForm(TForm1, Form1);
Application.Run;
end.
```

то Form2 станет главной Формой проекта. Вы также можете использовать эту страницу для определения условий - будет ли данная

Форма создаваться автоматически при старте программы (Auto - create forms) или только в процессе ее выполнения (Available forms). Если Форма создается не автоматически, а по ходу выполнения программы, то для этого нужно использовать процедуру Create (Создать).

Страница Applications

На странице Applications (Приложения), показанной на рис.27 вы можете задать Заголовок (Title), файл помощи (Help File) и Пиктограмму (Load Icon) для проекта.

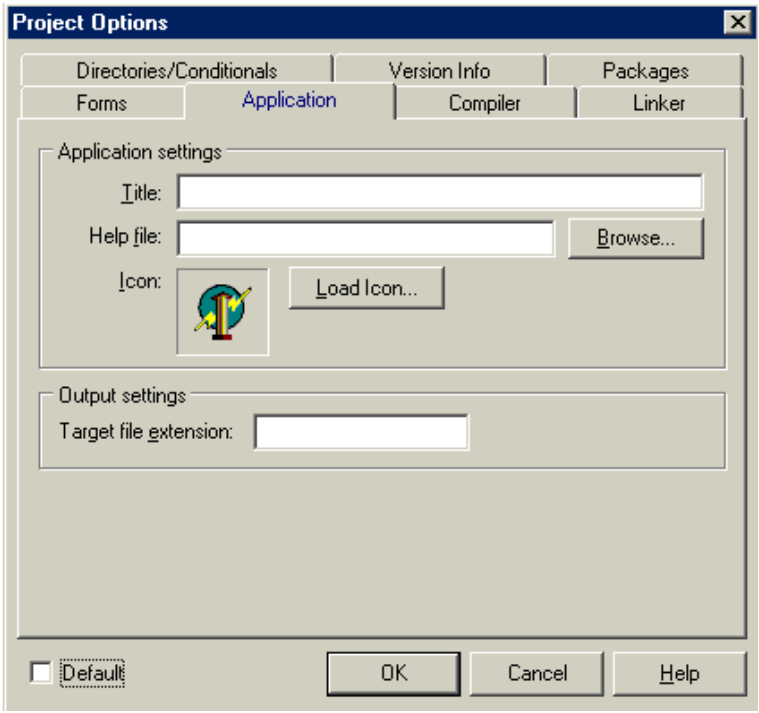

Рис.27. Страница общих установок для приложения.

С помощью последней опции Target file extension, можно переопределить стандартное расширение создаваемого файла (DLL - для динамически линкуемых библиотек, OCX - для элементов ActiveX и т.д.).

Однако это расширение можно не устанавливать, так как Delphi весьма корректно работает "по умолчанию".

Страница Compiler

Ранее уже говорилось, что установки из пункта меню Options, Project сохраняются в соответствующем файле с расширением OPT. Давайте рассмотрим теперь директивы (параметры) компилятора на странице Compiler, которая показана на рис.28.

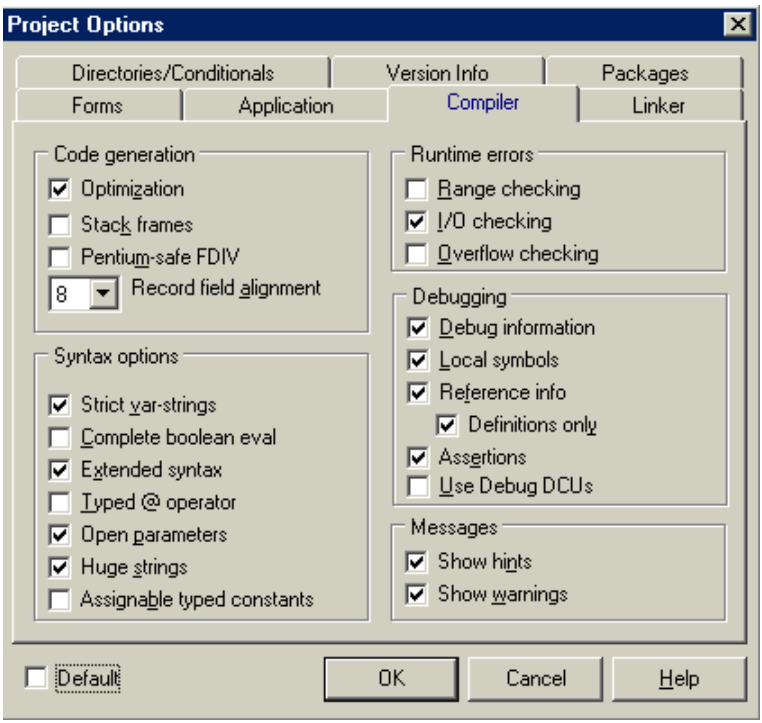

Рис.28. Страница для определения
директив компилятора.

Приведенная ниже таблица показывает, как некоторые директивы отображаются в OPT файле (столбец таблицы - OPT File), на странице Compiler (столбец таблицы - Options Page) и внутри кода самой программы (столбец таблицы - Editor Symbol).

OPT File	*Options Page*	*Editor Symbol*
U	Pentium - Safe FDIV	{$U + }
R	Range Checking	{$R + }
I	IO Checking	{$I + }
Q	Overflow Checking	{$Q + }
V	Strict Var Strings	{$V + }
B	Complete Boolean Eval	{$B + }
X	Extended Syntax	{$X + }
T	Typed @ Operator	{$T + }
P	Open Parameters	{$P + }
D	Debug Information	{$D + }
L	Local Symbols	{$L + }

Кроме того, опции Show hints и Show warnings, помогут вам при отладке (при этом компилятор будет выдавать множество предупреждений, например, об использовании неинициализированной переменной).

Страница Linker

Теперь перейдем к странице Linker (Отладчик), показанной на рис.29:

- Секция Memory sizes определяет минимальный и максимальный размеры стека приложения и устанавливает предпочтительный базовый адрес для вашей библиотеки DLL. Система Delphi сама устанавливает "по умолчанию" размеры Stack Size и Image Base.
- Раздел EXE Description предоставляет возможность внести в EXE или DLL строку описания приложения.
- Опция Map file полезна для тех программистов, которые интересуются технической информацией, например адресами сегментов, стартовым адресом программы и т.д.
- Linker output определяет, что именно будет выдавать компилятор - Delphi Compiled Unit (DCU) или объектные файлы (OBJ). Объектные файлы могут применяться, например, при разработке программ с использованием двух языков.
- Опции EXE и DLL позволяют создавать консольные приложения и включать отладочную информацию.

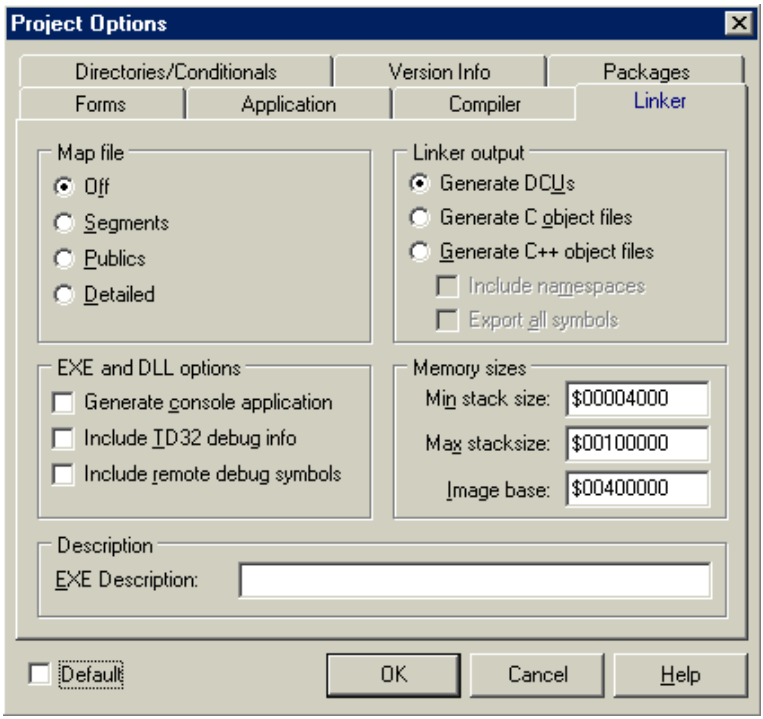

Рис.29. Страница линковщика.

Страница Directories/Conditionals

Страница Directories/Conditionals, показанная на рис.30 дает возможность расширить число директорий, в которых компилятор и линковщик ищут DCU файлы. В файле DELPHI.INI содержится еще один список директорий, с которыми может работать система Delphi. Запомните, что в OPT файле находится список директорий для конкретного проекта, а в файле DELPHI.INI этот список относится к любому проекту. Перечислим теперь некоторые из присутствующих на этой вкладке параметров:

- Output directory и Unit output directory - выходная директория, куда помещаются EXE и DCU файлы, получающиеся при компиляции программы. Если оставить опции незаполненными, создаваемые модули будут располагаться там же, где и исходные тексты программ, а вы-

ходные, выполняемые файлы или DLL библиотеки - в папке проекта.

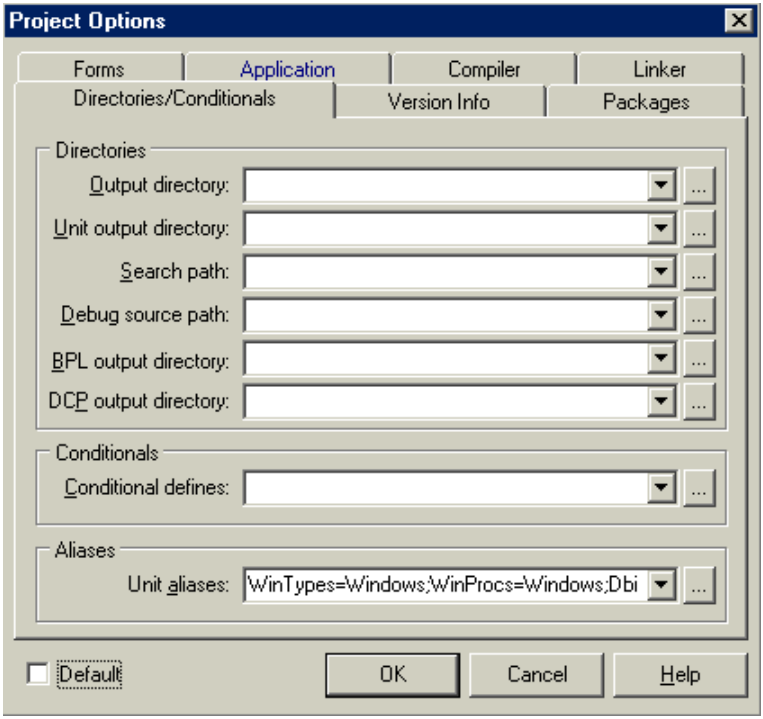

Рис.30. Страница Directories/Conditionals.

- Search path - список директорий для поиска DCU файлов при линковке. Директории перечисляются через точку с запятой ";".
- Conditional defines - для опытного программиста и на первом этапе создания проекта не требуется. Эти опции определяют флаги, проверяемые в процессе компиляции и используемые, как правило, для включения или исключения блоков кода из проекта при компиляции.
- Unit Aliases - существует для совместимости со старыми версиями Delphi.

Version Info

Следующая вкладка Version Info дает возможность добавить к выполняемому EXE модулю или DLL библиотеке информацию о версии -

Major Version, Minor Version и File Description.

Действительно полезную возможность предоставляет группа Auto - increment build number, заставляя Delphi всякий раз увеличивать номер выпуска при компиляции программы.

Раздел Module Attributes позволяет включать флаги в создаваемом приложении, такие как Debug Build. Выбор опций не влияет на процесс компиляции - они используются только в информативных целях.

Packages

Вкладка Packages позволяет определить, какие пакеты доступны для использования при разработке приложения, и прилинковать их при создании результирующего файла. Группа Design packages предоставляет список зарегистрированных пакетов, которые можно выбрать для использования в вашем приложении.

Группа Runtime packages дает возможность определить, какие пакеты компоновщик будет использовать при построении выходного файла. По умолчанию опция Build with runtime packages отключена, а это означает, что все объекты из VCL будут скомпонованы с вашим приложением. Включение опции означает, что ваше приложение будет разделять с другими приложениями Delphi одну копию пакетов.

ПАНЕЛИ ИНСТРУМЕНТОВ

Основная палитра компонентов Delphi имеет семнадцать страниц или вкладок. Рассмотрим основные панели более подробно и приведем общее описание некоторых других панелей инструментов.

Стандартная панель

На первой странице Палитры Компонент (Standard) размещены 17 объектов (рис.31), которые играют важную роль при разработке любого проекта. Мало кто обойдется без таких объектов, как кнопки, списки, окна ввода и т.д. Большинство компонентов на этой странице являются аналогами экранных элементов самой системы Windows. Компоненты Delphi обладают также некоторыми удобными дополнительными встроенными возможностям.

Набор и порядок компонент на каждой странице являются конфигурируемыми (пункт Главного меню Component). Так, вы можете добавить к имеющимся компонентам новые, изменить их количество и порядок. Это можно сделать, и вызвав всплывающее, Контекстное меню - для этого нужно нажать правую кнопку мыши, когда ее указатель находится на Палитре компонент, и выбрать пункт Properties (Свойства).

Рис.31. Стандартная Палитра Компонент.

Некоторые стандартные компоненты Delphi перечислены ниже, где даны также краткие комментарии по их применению. При изучении данных компонент полезно иметь перед собой компьютер, чтобы сразу посмотреть, как они работают и как ими можно пользоваться:

 Курсор - не компонент, просто пиктограмма для быстрого выбора какого - либо объекта.

 TMainMenu (Главное меню) - позволяет вам поместить Главное меню на Форму программы. При помещении TMainMenu на Форму оно выглядит просто, как иконка (Пиктограмма). Иконки данного типа называют невидимыми компонентом, поскольку они неви-

димы во время выполнения программы. Создание меню включает три шага:

1. Помещение TMainMenu на Форму - щелкните по кнопке компонента, а затем в нужном месте Формы.
2. Вызов Дизайнера Меню через свойство Items в Инспекторе Объектов.
3. Определение пунктов или разделов меню в Дизайнере Меню.

 TPopupMenu (Всплывающее меню) - позволяет создавать всплывающие, Контекстное меню. Этот тип меню появляется при щелчке правой кнопки мыши на любом объекте, к которому привязано данное меню. У всех видимых объектов имеется свойство PopupMenu, где и указывается нужное меню. Создается PopupMenu аналогично Главному меню.

 TLabel (Метка) - служит для отображения текста на Форме. Вы можете изменить шрифт и цвет метки, если дважды щелкнете по свойству Font (Шрифт) в Инспекторе Объектов. Это легко сделать и во время выполнения программы, написав всего одну дополнительную строчку кода, привязанного к такой метке.

 TEdit (Окно ввода в одну строчку) - стандартный управляющий элемент системы Windows для ввода информации. Он может быть использован для отображения короткого фрагмента текста и позволяет пользователю вводить текст во время выполнения программы.

 TMemo (Окно ввода из нескольких строк) - иная форма окна TEdit. Подразумевает работу с большими многострочными текстами. TMemo может переносить слова на следующую строку, сохранять в ClipBoard (Буферная память Windows) фрагменты текста, а также выполнять другие основные функции редактора текста. TMemo имеет ограничения на объем текста в 32Кб, что составляет около 10 - 20 страниц.

 TButton (Кнопка) - позволяет выполнить какие - либо дейст-

вия при нажатии кнопки мышкой во время выполнения программы. Поместив TButton на Форму, вы по двойному щелчку можете создать заготовку обработчика события при нажатии этой кнопки. Далее нужно заполнить заготовку некоторым кодом, например:

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

 TCheckBox (Переключатель типа и - и) - отображает строку текста с маленьким окошком рядом. В окошке можно поставить отметку (флажок), которая означает, что этот пункт выбран. Например, если посмотреть окно диалога настроек компилятора (пункт Главного меню Project, команда Options, страница Compiler), то можно увидеть, что оно состоит преимущественно из переключателей типа CheckBox.

 TRadioButton (Переключатель типа или - или) - позволяет выбрать только одну опцию из нескольких возможных. Если вы опять откроете диалог Project, Options и выберете страницу Linker, то сможете увидеть, что секции Map file и Link output состоят из наборов RadioButton.

 TListBox (Выбор объекта из списка) - нужен для показа прокручиваемого списка объектов. Классический пример ListBox для многих приложений в среде Windows это выбор файла из списка в пункте меню File, Open. Названия файлов или директорий находятся именно в списке ListBox.

 TComboBox (Ввод или выбор из списка) - во многом напоминает ListBox, за исключением того, что позволяет водить информацию в маленьком поле ввода сверху окна типа ListBox. Есть несколько типов ComboBox, но наиболее популярен - спадающий вниз (Drop - Down ComboBox), который можно видеть внизу окна диалога выбора файла (File, Open).

 TScrollbar (Полоса прокрутки) - полоса прокрутки, которая автоматически появляется в объектах редактирования и окнах ListBox при необходимости прокрутки текста для его просмотра.

 TGroupBox (Групповая панель) - используется для визуальных целей и для указания Windows, каков порядок перемещения по компонентам на Форме (при нажатии клавиши TAB).

 TRadioGroup (Группа объектов RadioButton) - используется аналогично TGroupBox, для группировки объектов TRadioButton.

 TPanel (Панель) - управляющий элемент, похожий на TGroupBox, который используется в декоративных целях. Чтобы использовать TPanel, просто поместите его на Форму, а затем разместите на нем другие компоненты, объекты системы. Теперь при перемещении TPanel будут передвигаться и все эти компоненты. TPanel используется также для создания линейки инструментов и окна статуса.

Если вам нужна дополнительная и более полная информация по любым объектам системы, выберите на некоторой Палитре объект и нажмите клавишу F1 - появится Справочник с полным описанием данного объекта.

Панель Additional

На странице Standard представлены управляющие элементы, использующиеся в системе Windows, а на странице Additional (Дополнительная), показанной на рис.32 размещены объекты, позволяющие создать более красивый пользовательский интерфейс программы. Например, компонент OutLine удобен для отображения информации с иерархической структурой, а удивительный объект MediaPlayer позволит вашим программам воспроизводить звук, музыку и видео. Данная страница также содержит компоненты, главное назначение которых - отображение графической информации. Компонент TImage загружает и отображает растровые изображения некоторых графических форматов, а компонент TShape, украсит ваши формы окружностями, квадратами и т.д.

Приведем теперь список основных компонент Панели Additional:

Рис.32. Вид панели Дополнительная.

TBitBtn - кнопка вроде TButton, однако на ней можно поместить картинку (Glyph). TBitBtn имеет несколько предопределенных типов (bkClose, bkOK и т.д.), при выборе которых кнопка принимает соответствующий вид.


TSpeedButton - кнопка для создания панели быстрого доступа к различным командам (типа SpeedBar). Пример таких кнопок это Панель SpeedBar слева от Палитры Компонент в среде Delphi. Обычно на данную кнопку помещается только картинка (Glyph).

TMaskEdit - аналог TEdit, но с возможностью форматированного ввода. Формат определяется в свойстве EditMask. В Редакторе свойств, для объекта EditMask есть заготовки некоторых форматов - даты, валюты и т.д.

TStringGrid - служит для представления текстовых данных в виде таблицы. Доступ к каждому элементу таблицы происходит через свойство Cell (Ячейка).


TDrawGrid - служит для представления данных любого типа в виде таблицы. Доступ к каждому элементу таблицы происходит через свойство CellRect.


TImage - отображает на Форме графическое изображение. Воспринимает графические форматы BMP, ICO, WMF. Если картинку подключить во время дизайна программы, то она прикрепляется к EXE файлу после компиляции.


TShape - служит для отображения на Форме простейших

графических объектов типа окружности, квадрата и т.д.

 TBevel - элемент для рельефного оформления интерфейса.

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

Панель Dialogs

На странице Dialogs - Диалоги (рис.33) представлены компоненты для вызова стандартных диалогов системы Windows, а их вешний вид зависит от используемой версии Windows. Объекты, представленные на данной странице невидимы во время выполнения программы и вызов диалогов происходит программно, например:

```
If OpenDialog1.Execute then
Image1.Picture.LoadFromFile(OpenDialog1.FileName);
```

Рис.33. Панель Диалог.

Теперь просто перечислим эти диалоги в порядке их появления (слева на право) на странице Dialogs:

- OpenDialog - выбрать файл.
- SaveDialog - сохранить файл.
- OpenPictureDialog - открыть картинку.
- SavePictureDialog - сохранить картинку.
- FontDialog - настроить шрифт.
- ColorDialog - выбор цвета.
- PrintDialog - печать.
- PrinterSetupDialog - настройка принтера.
- FindDialog - поиск строки.
- ReplaceDialog - поиск с заменой.

Еще Windows 3.1 ввела в употребление стандартные диалоговые

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

Панель System

Страница представляет набор компонент для доступа к некоторым системным сервисам (функциям) типа таймер, DDE и т.д. (рис.34). Не каждая потребность, связанная с обработкой файлов, может быть удовлетворена с помощью стандартных диалоговых окон. Поэтому страница System предоставляет дополнительную возможность комбинировать отдельные элементы, такие как списки дисков, каталогов и файлов. Страница System также содержит компоненты, обрабатывающие обмен высокого уровня между программами посредством OLE (Object Linking and Embedding). А компонент TTimer может генерировать события через определенные, заранее установленные промежутки времени.

Рис.34. Панель Система.

Перечислим некоторые из этих компонент:

 TTimer - таймер, событие OnTimer (Включить таймер) периодически вызывается через промежуток времени, указанный в свойстве Interval (Интервал). Период времени может составлять от 1 до 65535 мс.

 TPaintBox - место (окно) для рисования. В обработчике событий, связанных с мышкой, в TPaintBox передаются относительные координаты указателя мышки.

 TMediaPlayer - медиа - плеер служит для управления мультимедийными устройствами типа CD - ROM, MIDI и т.д. Плеер выпол-

нен в виде панели управления с кнопками управления Play, Stop, Record и т.д. Для воспроизведения может понадобиться, как соответствующее оборудование, так и программное обеспечение (ПО). Подключение устройств и установка ПО должна производиться в среде Windows.

Панель Win 3.1

Панель (страница) Win3.1 (рис.35), позволяет создавать объекты, характерные для системы Windows версии 3.1. На этой странице находятся компоненты Delphi 1.0, возможности которых перекрываются аналогичными компонентами новой системы Windows 95/98.

Приведем краткое описание некоторых компонент этой панели:

Рис.35. Панель Win 3.1 со старыми компонентами Windows.

 TTabSet - горизонтальные закладки (вкладки). Обычно используется вместе с TNoteBook для создания многостраничных окон, примером которых служит сама Панель компонент. Название страниц можно задать в свойстве Tabs Инспектора объектов. Но проще это сделать в программе при создании Формы (OnCreate):

```
TabSet1.Tabs: = Notebook1.Pages;
```

Для того чтобы при выборе закладки страницы перелистывались нужно в обработчике события OnClick для TTabSet написать:

```
Notebook1.PageIndex: = TabSet1.TabIndex;
```

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

 TTabbedNotebook - многостраничный диалог со встроенными закладками, в данном случае закладки располагаются вверху окна.

 TNoteBook - используется для создания многостраничного диалога, причем на каждой странице располагается свой набор объектов. Используется совместно с объектом TTabSet, описанным выше.

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

 TFileListBox - специализированный объект типа ListBox, в котором отображаются файлы из указанной директории (свойство Directory). На названия файлов можно наложить маску - для этого служит свойство Mask (Маска). Кроме того, в свойстве FileEdit можно указать объект TEdit для редактирования маски.

 TDirectoryListBox - специализированный компонент типа ListBox, в котором отображается структура директорий текущего диска. В свойстве FileList можно указать TFileListBox, который будет автоматически отслеживать переход в другую директорию.

Другие Панели

Win32. Эта страница содержит компоненты, позволяющие Delphi программам использовать такие нововведения в пользовательском интерфейсе Windows, как просмотр древовидных структур и списков, панель состояния, присутствующая в интерфейсе программы Windows Explorer (Проводник), расширенный текстовый редактор и т.д.

Панели ***Data Access*** и ***Data Controls.*** Delphi использует механизм баз данных компании Borland (Borland Database Engine, BDE) для организации доступа к файлам баз данных различных форматов. Компоненты этих двух страниц облегчают программам Delphi использование сервиса баз данных, предоставляемого BDE, например многопользовательского считывания, записи, индексации и выдачи запросов для таблиц dBASE и Paradox. С использованием этих компонентов создание программы просмотра и редактирования базы данных почти не требует программирования.

Internet. Эта страница предоставляет компоненты для разработки приложений, позволяющих создавать HTML - файлы непосредственно из файлов баз данных и других типов, взаимодействующих с другими

приложениями для Internet. Delphi дает возможность создавать приложения для Web - сервера в виде DLL - файлов (Dynamic Link Library - Динамически компонуемая библиотека), способных содержать не визуальные компоненты. С помощью компонентов страницы Internet довольно просто создавать обработчики событий для обращения к определенному URL (Uniform Resource Locator - Унифицированный локатор ресурса), представлению документов в HTML - формате и пересылки их клиент - программе.

Samples. Эта, отличающаяся полнотой страница, содержит компоненты, которые не встроены в Delphi, но демонстрируют мощь системы компонентов. Для этих компонентов нет встроенной интерактивной справки. Все же они не менее полезны, чем компоненты с других страниц.

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

Использование Панелей

Размещать компоненты на форме очень просто - требуется только щелкнуть на нужной вкладке палитры компонентов, затем на кнопке с пиктограммой соответствующего компонента и после этого, щелкнуть в нужном месте окна Формы. Или можно щелкнуть на компоненте, а затем нарисовать на Форме прямоугольник с помощью мыши - компонент появится внутри этого прямоугольника. Если размеры компонента поддаются изменению, то при появлении на Форме он заполнит собой весь этот прямоугольник. Если вы забыли, на какой странице расположен конкретный компонент, выберите пункт Component List (Компоненты) из меню View (Вид), и на экране появится список компонентов в алфавитном порядке.

Если щелкнуть на компоненте в палитре компонентов, его кнопка окажется нажатой. Если щелкнуть на другом компоненте, первая кнопка вернется в исходное состояние - только один компонент может быть выбран в один момент времени. Для того чтобы все кнопки оказались в исходном состоянии, и было восстановлено нормальное использование мыши, следует щелкнуть на кнопке со стрелкой выбора, которая имеется с левой стороны каждой страницы палитры.

Рассмотрим теперь некоторый стандартный набор действий, необходимых для размещения компонента на Форме:

1. Выберите, например, страницу Standard палитры компонентов, щелкнув по ее заголовку мышкой.
2. Поместите указатель мыши на компонент с изображением

кнопки. После того, как мышь побудет в недвижимом состоянии секунду или две, появится поле помощи, идентифицирующее этот компонент, как кнопка Button.

3. Щелкните на кнопке, а затем щелкните на вашей Форме. Появится кнопка среднего размера, озаглавленная Button1.

4. Щелкните на компоненте Button опять, но на этот раз, удерживая левую кнопку мыши, нарисуйте прямоугольник большого размера прямо на Форме. Текст Button2 заполнит весь прямоугольник.

5. Обратите внимание, что кнопка имеет небольшие квадратные маркеры (угловые и боковые), позволяющие изменять ее размер. Используйте их, чтобы с помощью мыши изменить размеры кнопки.

6. Теперь щелкните мышкой на кнопке Button1 и перетащите ее в новое место.

7. Изменить положение объекта можно и кнопками со стрелками управления курсором при нажатой клавише Ctrl.

8. Изменить размер компонента можно теми же кнопками при нажатой клавише Shift.

При перемещении и изменении размера компоненты выравниваются по точкам координатной сетки Формы. Как правило, это хорошо - такая возможность помогает поддерживать порядок в формах. Если вы захотите отменить эту возможность или изменить плотность точек координатной сетки, выберите пункт Environment Options из меню Tools.

На странице Designer (Конструктор), флажки опций Display grid (Отображение сетки) и Snap to grid (Привязка к сетке) определяют, видна ли координатная сетка и активна ли она. Можно также изменить значения параметров Grid Size X (Шаг по оси X) и Grid Size Y (Шаг по оси Y), что приведет к изменению шага координатной сетки по горизонтали и вертикали, соответственно. Для лучшего управления размещением и размерами своих компонентов попробуйте следующее. Установите значение шага координатной сетки равным 4 вместо 8, отключите изображение координатной сетки, но оставьте включенным параметр Snap To grid.

Не каждый компонент виден на форме во время запуска программы. Например, размещение на форме компонента MainMenu приводит к появлению в разрабатываемом приложении меню, но соответствующая пиктограмма после запуска программы не отображается. Компоненты, представляющие диалоговые окна общего назначения, вообще никак не визуализируются во время работы программы. Размеры невидимого компонента в процессе разработки не изменяются - он всегда отображается в виде пиктограммы.

ФОРМА ПРОЕКТА

Чаще всего Delphi используется для создания различного рода приложений - прикладных пользовательских программ. Вы можете создавать приложения любого типа - от утилит командной строки до программы электронной почты или многопользовательской финансовой базы данных. В этой и следующих главах вы познакомитесь с методами созданием приложений, и с двумя основными типами приложений.

Работа с Формами

Подобно фундаменту здания, Форма представляет собой фундамент программы, на котором строится все остальное. Форма - это место, где пользователь общается с разрабатываемой программой и с самой средой Delphi. Любое приложение может иметь несколько Форм, каждая из которых выполняет свое, особое предназначение. Форма, создается с помощью конструктора Форм (Form Designer) и наследует основные Свойства, Методы и События этого класса.

Определим теперь базовое понятие объектно - ориентированного программирования - класс. Класс это категория (группа) объектов (компонент), обладающих одинаковыми Свойствами и поведением (откликом на События). При этом конкретный объект или компонент представляет собой просто экземпляр, какого - либо класса.

Свойства TForm

Класс TForm предоставляет возможность изменять его поведение и внешний вид Формы с помощью ряда Свойств, которые мы сейчас и рассмотрим.

Active

Свойство Active определяет, имеет ли Форма фокус, т.е. активна ли в данный момент и находится на переднем плане проекта. Если имеет, Свойство возвращает значение True, если нет - False. Windows выводит заголовок активной Формы цветом, отличающимся от цвета не активных окон.

Независимо от типа приложения, в один момент времени может быть активной только одна Форма. Вы можете обратить внимание на то, что заголовок родительской Формы в MDI (Multi Document

Interface) - приложении (см. далее) изображен "активным", обычно синим цветом.

ActiveControl

Свойство ActiveControl указывает на потомка TWinControl (который является родительским классом), имеющего в настоящий момент фокус ввода (активен). Вы можете установить значение этого Свойства во время создания программы, определив, какой элемент будет иметь фокус ввода при инициализации (запуске) Формы.

Назначение ActiveControl во время работы программы - установка фокуса ввода в поле с некорректно введенными данными. Приведенный ниже фрагмент кода позволяет проверить текст, введенный в элемент EditCustName, перед закрытием Формы:

```
procedure TDataEntryForm.FormCloseQuery (Sender: TObject;
var CanClose: Boolean);
begin
{Проверяем, введен ли текст в элемент}
If EdtCustName.Text = ' ' then
begin
{Запрещаем закрытие}
CanClose:= False;
{Устанавливаем фокус в поле с некорректными данными}
ActiveControl:= EditCustName;
end;
end;
```

Метод SetFocus устанавливает фокус ввода (делает активным) и обновляет Свойство ActiveControl. Большинство Событий объектов Формы передает параметр Sender. Этот параметр определяет, какой элемент обнаружил Событие и запустил его обработчика.

AutoScroll, HorzScrollBar и VertScrollBar

Свойство *AutoScroll* управляет появлением полос прокрутки на Форме, размеры которой не позволяют вывести все ее элементы одновременно. Если Свойство равно True, и вы изменили размеры так, что не все элементы Формы видны, на Форме автоматически появляются полосы прокрутки. Если же значение Свойства - False, вы теряете доступ к элементам Формы, не поместившимся на экране. Компонент

TScrollBar, позволяет прокручивать Форму независимо от Свойства AutoScroll.

Вы можете управлять полосами прокрутки с помощью Свойств HorzScrollBar и VertScrollBar. Это бывает полезно, поскольку, например, размеры выводимой диаграммы могут быть больше размеров Формы, а выводите вы ее самостоятельно, и Свойство AutoScroll не всегда активизируется. Дадим пример кода, обеспечивающего прокрутку в двойном размере Формы:

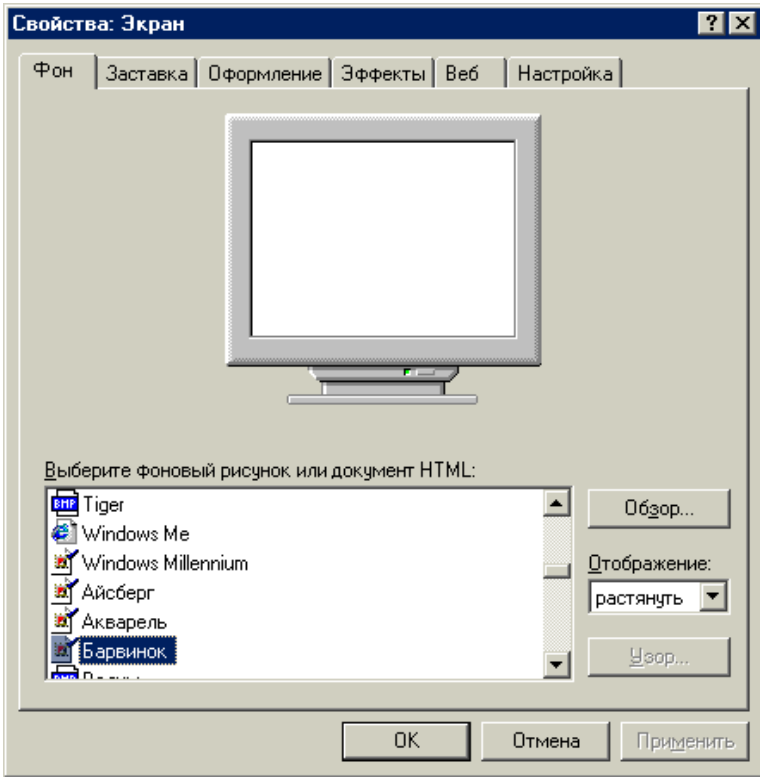

Рис.36. Доступ к контекстно - зависимой справке.

```
{Устанавливаем диапазон вертикальной прокрутки}
VetrScrollBar.Range:= Height * 2;
{Показываем вертикальную полосу прокрутки}
VertScrollBar.Visible:= True;
```

```
{Устанавливаем диапазон горизонтальной прокрутки}
HorzScrollBar.Range:= Width * 2;
{Показываем горизонтальную полосу прокрутки}
HorzScrollBar.Visible:= True;
```

BorderIcons

Свойство Borderlcons представляет собой набор логических значений, использующийся для определения набора пиктограмм в заголовке Формы (Синяя полоса вверху Формы). Значения biMinimize и biMaximize создают пиктограммы (кнопки), которые позволяют свернуть и развернуть Форму с помощью мыши. Для того чтобы значения biMinimize и biMaximize работали, необходимо установить Свойство BorderStyle равным bsSizeable или bsSizeToolWin.

Значение biHelp выводит на Форме кнопку с вопросительным знаком. Щелчок на ней вызывает контекстно - зависимую справку, которая выводится, как текст подсказки Hint, т.е. без вызова Windows Help. Пример такой кнопки показан справа вверху окна на рис.36.

Значение biSystemMenu создает в левой части заголовка пиктограмму, позволяющую вызывать системное меню, как показано на рис.37. Для того чтобы значения biMinimize, biMaximize и biHelp работали, необходимо присвоить Свойству BorderIcons значение biSystemMenu.

BorderStyle

Свойство BorderStyle имеет перечислимый тип и позволяет определить:

- Вид заголовка Формы.
- Доступные кнопки в заголовке Формы.
- Отображение строки меню.
- Поведение границ Формы.

На рис.38 показана Форма для шести значений BorderStyle. Каждая Форма создавалась, как Форма размером 200x200 пикселей. По умолчанию Свойство BorderStyle имеет значение bsSizeable, создающее обычное окно с изменяемыми размерами. Такое окно имеет стандартную строку заголовка и не имеет ограничений по расположению на ней кнопок. Примеры таких окон - программы Explorer (Проводник) и Notepad (Блокнот).

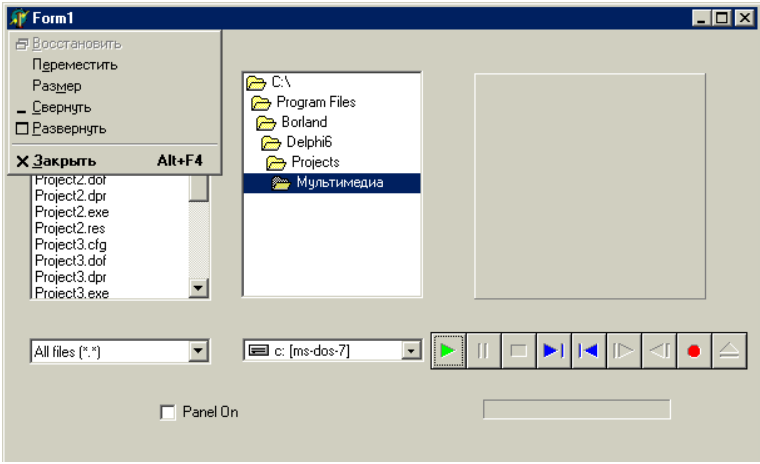

Рис.37. Системное меню позволяет перемещать и закрывать Форму.

Значение bsDialog создает диалоговое окно, которое используется, когда проект требует от вас ответа для продолжения выполнения программы или для вывода информации. Видимое различие между стандартными (bsSizeable) и диалоговыми окнами, в связи с тем, что последнее не может изменять размеры, состоит лишь в том, что указатель мыши не изменяется при пересечении рамки, границы окна. Значения biMinimize и biMaximize Свойства BorderIcons не будут работать, если Свойство BorderStyle установлено равным bsDialog.

Третий по популярности стиль окна - bsSingle создает Форму, которая не может изменять размеры во время работы. В отличие от bsDialog, bsSingle не запрещает установку любых пиктограмм. Единственное ограничение состоит в том, что кнопка Развернуть, будучи выведенной, является недоступной (блокированной). Одним из примеров такой программы является Calculator (Калькулятор), окно которого показано на рис.39.

Панель инструментов (ToolBar) позволяет быстро получить доступ к сгруппированным стандартным функциям. В Delphi можно сконструировать панель инструментов, поместив группу компонентов TSpeedButton на Форму, имеющую стиль bsSizeToolWin или bsToolWindow.

Окно в стиле bsSizeToolWin окно может изменять размеры, но не имеет кнопок biMinimize, biMaximize HbiHelp. Окно в стиле

bsToolWindow действует так же, но не позволяет изменять размеры. Стиль bsNone создает окно без рамки и заголовка. Такое окно не рекомендуется использовать в качестве стандартного или диалогового, однако оно может быть полезным в программах сохранения экрана или заставках.

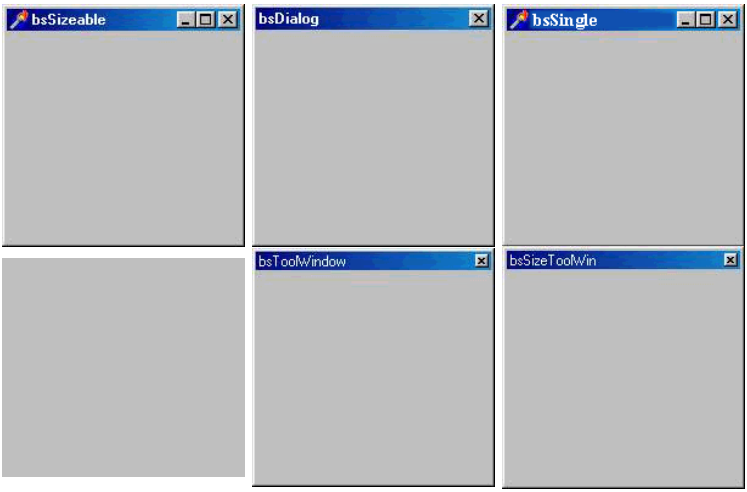

Рис.38. Влияние значения BorderStyle на вид Формы. Значения по часовой стрелке, начиная с левой верхней Формы, таковы: bsSizeable, bsDialog, bsSingle, bsNone, bsToolWindow и bsSizeToolWin.

Если вы выбрали для Свойства BorderStyle значение, создающее окно с разрешенным изменением размеров, Delphi автоматически установит значение AutoScroll равным True. Формы со стилями bsDialog и bsNone не могут иметь строки Главного меню.

Height и Width

Эти Свойства определяют высоту и ширину Формы в пикселях и обычно используются для изменения размеров Формы во время работы на дисплеях разной разрешающей способности. Приведем пример увеличения размеров Формы до размеров всего экрана:

```
{Перемещаем Форму в верхний левый угол экрана}
Left:= 0;
```

```
Top:= 0;
```

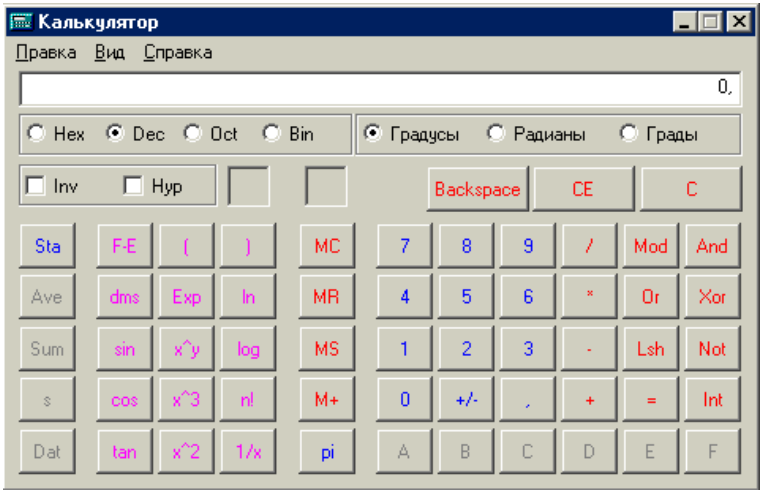

Рис.39. Стиль bsSingle полезен, когда пользователю не надо изменять размер окна.

```
{Изменяем размеры Формы}
Width:= Screen.Width;
Height:= Screen.Height;
```

Класс TScreen, о котором будет сказано ниже, и его экземпляр Screen, предоставляют доступ к информации о размерах всего экрана. Приведенный код, конечно, работает, но плохо, так как требуется четыре обновления Формы. На самом деле лучше использовать Метод SetBounds:

```
SetBounds(0, 0, Screen.Width, Screen.Height);
```

ClientHeight и ClientWidth

Любое окно состоит из двух частей - клиентской и не клиентской. Обычно приложение выводит изображения только в клиентской области окна, размер которой возвращается через Свойства ClientHeight и ClientWidth. Эти Свойства используются для того, чтобы убедиться, что на Форме может выводиться весь объект определенного размера.

Показанный ниже текст приводит размер клиентской области

Формы в соответствие размерам изображения, содержащегося на компоненте типа Timage или ImgPicture:

```
with imgPicture.Picture do
begin
{Изменение размера}
ClientWidth:= Width;
ClientHeight:= Height;
end;
```

Не клиентская область окна обычно прорисовывается самой системой Windows и включает строку заголовка, меню и рамку окна.

FormStyle

Свойство FormStyle имеет перечислимый тип и определяет, как Форма взаимодействует с вашим приложением и системой Windows. Существует два основных стиля Форм - MDI (Multiple Document Interface - многодокументный интерфейс) и не MDI (SDI - см. далее). Имеется два MDI - стиля (fsMDIForm и fsMDIChild), которые также будут рассмотрены ниже. Не MDI Формы существуют также в двух вариантах - fsNormal и fsStayOnTop. Наиболее популярен стиль fsNormal, который создает стандартный стиль, используемый для диалогов, панелей инструментов и SDI - приложений. Стиль fsStayOnTop применяется реже и создает Форму, всегда остающуюся поверх других Форм и приложений. Это может быть полезно при выводе системной информации и использовании ресурсов компьютера.

Примером такого окна является окно программы Chat, используемой при работе в сети и Интернет. Вот как можно реализовать, подобно программе Chat, установку вывода данного окна поверх других окон путем выбора пункта меню:

```
procedure TFormI.mnuAlwaysOnTopClick (Sender: TObject);
begin
with mnuAlwaysOnTop do
begin
{Переключаем отметку выбора пункта меню}
Checked:= not Checked;
{Проверка установок меню}
If Checked then
{Устанавливаем стиль fsStayOnTop.}
```

```
FormStyle:= fsStayOnTop
else
{Возвращаем нормальный стиль}
FormStyle:= fsNormal;
end;
end;
```

Изменение Свойства FormStyle вызывает Событие OnShow - показывать Форму.

Icon

Свойство Icon определяет пиктограмму, выводимую Windows при сворачивании вашей Формы. В интерфейсе Windows 95/98 эта пиктограмма выводится также в левом верхнем углу Формы на кнопке системного меню. Если вы не определите значения для этого Свойства, будет использоваться Свойство Icon глобального объекта Application.

KeyPreview

Объект TForm наследует от класса TWinControl обработку Событий OnKeyDown (клавиша внизу), OnKeyUp (клавиша вверху) и OnKeyPress (клавиша нажата), и Свойство KeyPreview определяет ситуации, в которых эти сообщения запускаются. Если значение KeyPreview равно False, События клавиатуры пересылаются только тому управляющему элементу, который имеет фокус ввода (активен). Если же значение этого Свойства установлено равным True, Событие сначала пересылается Форме, а затем - управляющему элементу, имеющему фокус ввода.

Обычно Свойство KeyPreview используется для обработки функциональных клавиш, которые должны быть переданы Форме независимо от текущего активного элемента. Без этого Свойства обработка функциональных клавиш сводится к написанию для каждого элемента дополнительного обработчика, который должен отслеживать нажатие функциональных клавиш.

При установке Свойства KeyPreview, равным True все нажатия клавиш отсылаются обработчикам Событий OnKeyDown, OnKeyUp и OnKeyPress автоматически и для "отлова" положения функциональной клавиши надо написать только один обработчик События OnKeyDown Формы.

Приведем пример закрытия Формы при нажатии клавиши F2:

```
procedure TForml.FormKeyDown (Sender: TObject; var Key:
Word; Shift: TShiftState);
begin
{Проверить, нажата ли клавиша F2}
If Key = VK_F2 then
{Закрыть Форму}
Close;
end;
```

Рассматривая принцип работы этого кода, имейте в виду, что События OnKeyDown и OnKeyUp используют виртуальные коды.

Menu

Это Свойство определяет компонент TMainMenu, который дает возможность включить Главное меню Формы и позволяет сделать меню контекстно - зависимым. На рис.40 показано окно редактора WordPad с примером такого меню.

Для изменения Свойства Menu просто присвойте ему новое значение:

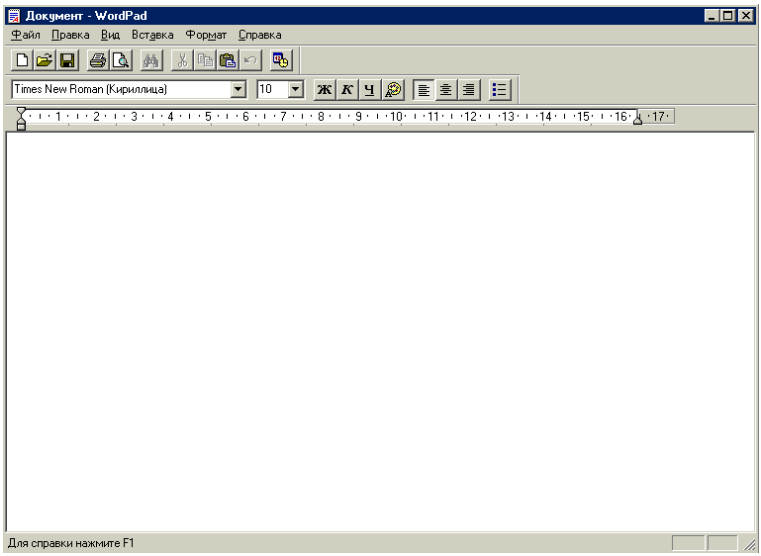

Рис.40. Можно создать контекстно - зависимое меню с помощью Свойства Menu.

```
Menu:= mnuMainMenu;
```

Position

Position - это Свойство перечислимого типа, определяющее размещение Формы при запуске приложения. Значение по умолчанию (poDesigned) заставляет Форму выводиться в месте, определенном при разработке приложения. Положение и размер Формы при этом берутся из Свойств Left, Top, Height и Width. Поскольку вы не можете знать заранее, в какой системе будет запущено ваше приложение, может оказаться, что на мониторе с низким разрешением при использовании этого значения Свойства будет видна только часть Формы. Более полезно значение poScreenCenter, использующее заданные вами при создании приложения значения Width и Height, но изменяющее Свойства Left и Top так, что Форма выводится в центре экрана.

Если вы установите Position равным poDefault, Windows автоматически установит размеры и положение Формы, но вы лишитесь возможности контролировать ее размеры. При таком значении Свойства, можно создать Форму размером 200x200, которая будет выведена на экран как 640x480. Из - за этого, в частности, не допускается применение данного значения для MDI - Форм. Значение poDefaultPosOnly более полезно, так как оно автоматически определяет положение Формы, но не ее размеры (а потому рекомендуется для MDI - Форм, в которых требуются определенные размеры дочерних Форм). Последнее значение Свойства (poDefaultSizeOnly) автоматически определяет размер, но не расположение Формы. Это значение может использоваться там, где важно положение Формы на экране, а не ее реальный размер.

Хотя Свойство Position позволяет определить, каким образом будет выводиться Форма, профессионально сделанные приложения сами запоминают свое расположение на экране и при следующем запуске выводятся в той же позиции и с тем же размером. Это осуществимо, например, благодаря записи положения окна в Registry Windows или INI - файл, который должен находиться в том же каталоге, где расположено само приложение. Обычно сохранение позиции и размеров экрана выполняется в самый последний момент - при закрытии Формы:

```
procedure TForm1 .FormClose(Sender: TObject);
var
sAppPath: String;
iniSettings: TINIFile;
```

```
begin
{Получить путь к EXE - файлу приложения}
sAppPath:= ExtractFilePath(Application.EXEName);
{Создаем объект TINIFile}
iniSettings:= TINIFile.Create(sAppPath + 'SETTINGS.INI');
try
{Записываем Свойства в INI - файл}
iniSettings.Writeinteger(Name,'Left',Left);
iniSettings.Writeinteger(Name,'Top',Top);
iniSettings.Writeinteger(Name,'Width',Width);
iniSettings.Writeinteger(Name,'Height',Height);
finally
iniSettings.Free;
end;
end;
```

Этот код записывается в Событие onClose Инспектора объектов вашей Формы. Для компиляции примера не забудьте включить в раздел uses своего модуля внешний модуль INIFiles. После выполнения приведенного выше кода, ваш INI - файл будет содержать запись вида:

```
[Form1]
Left=108
Top=174
Width=540
Height=165
```

Заметьте, что для строки раздела используется Свойство Name вашей Формы. Восстановить положение Формы на экране немного сложнее (главным образом из - за того, что следует отработать ситуацию, когда INI - файла нет):

```
procedure TForm1.FormCreate(Sender: TObject);
const
CNOTFOUND = - 1;
var
sAppPath: String;
iniSettings: TINIFile;
liValue: Longint;
begin
{Получаем путь к EXE - файлу приложения}
```

```
sAppPath:= ExtractFilePath(Application.ExeName);
{Создаем объект TINIFile}
iniSettings:= TINIFile.Create(sAppPath + 'SETTINGS.INI');
try
{Пытаемся считать значение Left}
liValue:= iniSettings.Readinteger (Name, 'Left', cNOTFOUND);
{Проверяем, считано ли значение}
if liValue = cNOTFOUND then
begin
{Свойства в INI - файле не найдены - центруем Форму}
Left:= (Screen.Width - Width) div 2;
Top:= (Screen.Height - Height) div 2;
end
else
begin
{Считываем значения из INI - файла}
Left:= iniSettings.Readinteger(Name,'Left',Left);
Top:= iniSettings.Readinteger(Name,'Top',Top);
Height:= iniSettings.Readinteger(Name,'Height',Height);
Width:= iniSettings.Readinteger(Name,'Width',Width);
end;
finally
iniSettings.Free;
end;
end;
```

Этот код записывается в Событие onCreate Инспектора объектов запускаемой Формы.

WindowState

Свойство перечислимого типа WindowState определяет состояние окна - свернутое, развернутое или нормальное. По умолчанию оно имеет значение wsNormal (при этом окно выводится в состоянии, определяемом Свойствами Position, Left, Top, Height и Width). Чтобы свернуть или развернуть Форму, используются значения wsMinimize и wsMaximize.

События TForm

Класс TForm добавляет несколько Событий к родительскому

классу TWinControl. Эти События позволяют изменять поведение Формы путем выполнения загрузки и сохранения информации о состоянии Формы или распределения и освобождения дополнительных ресурсов. Когда Форма создается и отображается, происходят следующие События:

- OnCreate - запускается при создании, открытии Формы и позволяет распределять ресурсы компьютера и инициализировать саму Форму.

- OnShow - происходит непосредственно перед выводом Формы на экран. К этому времени все элементы управления и компоненты созданы и инициализированы. Хотя к тому моменту, когда происходит Событие OnShow, на экране Форма еще не видна, Свойство Visible должно быть установлено равным True.

- OnResize - генерируется при изменении размера Формы во время выполнения приложения. Обычно здесь помещается код для изменения размера и положения на экране элементов управления, не поддерживающих Свойство Align. Событие OnResize генерируется также при создании Формы, когда Delphi устанавливает ее начальные размеры. OnResize вызывается и в процессе изменения размеров Формы.

- OnActivate - происходит при получении Формой фокуса ввода. OnActivate вызывается только при переходе фокуса ввода от одной Формы к другой в пределах одного приложения. При переключении между приложениями, Delphi генерирует Событие OnActivate глобального объекта Application.

- OnPaint - запускается, когда необходимо перерисовать Форму. Это может происходить, когда Форма только что стала видимой, при частичном удалении перекрывающих ее элементов или увеличении размеров. Событие полезно, если вы перерисовываете какую - то часть Формы самостоятельно.

- Событие OnCreate происходит один раз за все время существования Формы, прочие же События могут вызываться неоднократно. При закрытии и уничтожении Формы, также генерируются некоторые События:

- OnCloseQuery - генерируется в ответ на действия, закрывающие Форму. Обработчик Событий получает логическую переменную CanClose, определяющую, может ли Форма быть закрыта. По умолчанию она имеет значение True, но если вы в обработчике Событий установите False, Форма останется открытой. Обычно это используется для сохранения, не сохраненных файлов или для подтверждения закрытия Формы. Приведем пример такого кода:

```
procedure TForm1.FormCloseQuery (Sender: TObject;
var CanClose: Boolean);
begin
CanClose:= MessageDlg('Close form?', mtConfirmation,
[mbYes,mbNo], 0) = mrYes;
end;
```

- OnClose - генерируется непосредственно перед закрытием Формы. Обычно это Событие используется для изменения стандартного поведения Формы при ее закрытии. Для этого Delphi передает в обработчик События переменную Action, которая может принимать одно из четырех значений: caHide, caMinimize, caNone или caFree. По умолчанию для не MDI - Форм используется значение caHide, скрывающее Форму. Для дочерних MDI - Форм, используется значение "по умолчанию", сворачивающее Форму, которое равно caMinimize. Если Action установлено равным caNone, закрытия не происходит. Значение caFree заставляет Delphi закрыть Форму и освободить всю связанную с ней память.

- Событие OnClose вызывается только при закрытии Формы с помощью щелчка мышкой на кнопке закрытия или вызова функции Close. Если вы закрываете главную Форму приложения, все другие открытые Формы закрываются без вызова События OnClose. Событие OnCloseQuery вызывается всегда, независимо от способа закрытия Формы.

- OnDeActivate - происходит при потере Формой фокуса ввода. Запуск происходит по тем же правилам, что и запуск События OnActivate.

- OnHide - запускается непосредственно перед тем, как Форма станет невидимой. Хотя при вызове OnHide Форма еще видна, ее Свойство Visible устанавливается равным False.

- OnDestroy - генерируется непосредственно перед уничтожением Формы. Обычно оно используется для освобождения ресурсов, выделенных после События OnCreate.

Событие OnDestroy вызывается только один раз за все время существования Формы, все прочие События могут вызываться неоднократно.

Многократное использование Форм

К этому моменту вы уже должны быть хорошо знакомы с объект-

но - ориентированной природой Delphi. Поскольку TForm представляет собой класс, он может повторно использоваться, расширяться и изменяться. Повторное и многократное применение Форм поддерживается через шаблоны Форм и наследование. Оба эти Метода используют Object Repository - хранилище объектов.

Шаблоны Форм

Шаблоны Форм (Form Templates) предоставляют основу для любой новой Формы и по одной заготовке (шаблону) можно создать несколько различных Форм. В Delphi есть хранилище объектов (Object Repository), в котором содержится множество различных шаблонов Форм, всевозможных диалогов и готовых проектов.

Приведенные ниже опции позволяют использовать шаблоны Форм:

- Copy - опция добавляет копию шаблона Формы в ваш проект. Изменения объекта в проекте не влияют на объекты хранилища.
- Use - опция связывает шаблон непосредственно с вашим проектом. Изменения в проекте воздействуют на объект, находящийся в хранилище, и наоборот.

Для иллюстрации сказанного добавим в проект новую Форму, основанную на шаблоне About box, следующим образом:

1. Выберите команды File, New, Application. Появится пустое приложение.
2. Выберите команду File, New, Other. Появится диалоговое окно New Items.
3. Щелкните на вкладке Forms вверху диалогового окна - Delphi выведет доступные шаблоны Форм, как показано на рис.41.
4. Убедитесь, что выбран Метод Copy или Use.
5. Щелкните на кнопке OK - новая Форма About box будет добавлена в ваш проект.

Если вы выбрали опцию Copy, новая Форма будет дубликатом шаблона и дальнейшая работа с ней не отразится на шаблоне - оригинале, который находится в хранилище. При использовании опции Use все изменения в одном проекте через хранилище Форм будут переданы во все проекты, применяющие эту Форму с опцией Use.

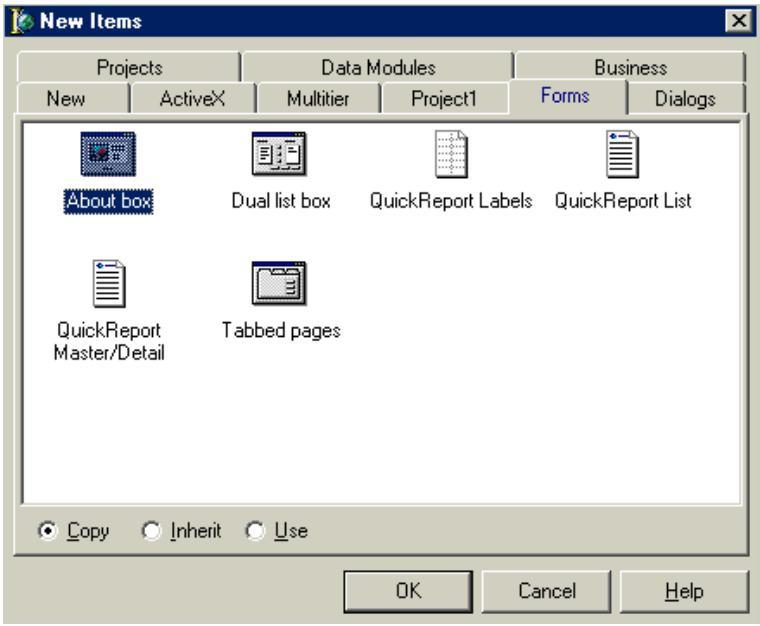

Рис.41. Шаблоны Форм позволяют использовать заготовленные общие Формы.

Добавление собственного шаблона

Хотя имеющиеся шаблоны весьма полезны и хорошо сделаны, для продолжительной профессиональной работы их будет недостаточно. Наверняка, нужно будет что - либо доработать или создать новую Форму, которую можно было бы использовать в других приложениях. Чтобы не делать одну и ту же работу дважды (трижды и т.д.), создайте шаблон Формы и поместите его в хранилище объектов. Для этого выполните следующие действия:

1. Создайте Форму, добавьте в нее компоненты и введите нужные коды. Конечно, вы вполне можете использовать в качестве заготовки имеющийся шаблон.
2. Сохраните Форму в папке OBJREPOS Delphi. Object Repository не хранит копий всех шаблонов, а устанавливает связи с их DFM и PAS - файлами. Если вы удалите такой файл, шаблон станет недееспособ-

ным.

3. Щелкните правой кнопкой мыши на Форме и выберите команду Add to Repository. При этом Delphi выведет диалоговое окно Add To Repository, показанное на рис.42.

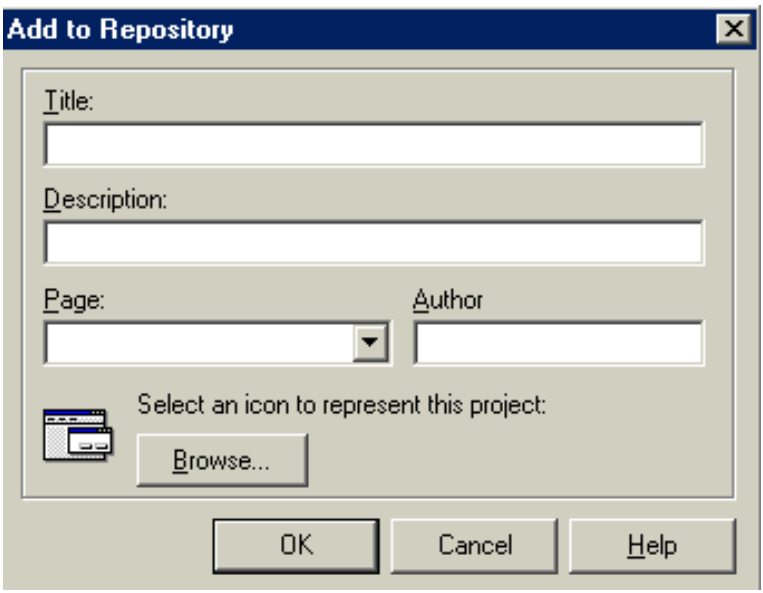

Рис.42. Используйте диалоговое окно Add To Repository для добавления ваших собственных шаблонов в хранилище объектов.

4. Введите заголовок Формы в поле Title (он будет использоваться в качестве подписи под пиктограммой в диалоговом окне New Items).

5. Из раскрывающегося списка Page выберите страницу, на которой будет размещен шаблон.

6. Щелкните на кнопке OK, и Форма будет добавлена в Object Repository.

Delphi автоматически включает вновь созданные Формы в текущий проект. Они связаны с проектом, как будто вы воспользовались опцией Use. Вам может не понравиться то, что ваши шаблоны содержатся в хранилище Delphi. Предположим, что вы де инсталлируете Delphi (при этом, чтобы сохранить свои шаблоны, вы будете вынуждены вручную сохранять их в другом месте, а потом вновь добавлять их в

очередное хранилище).

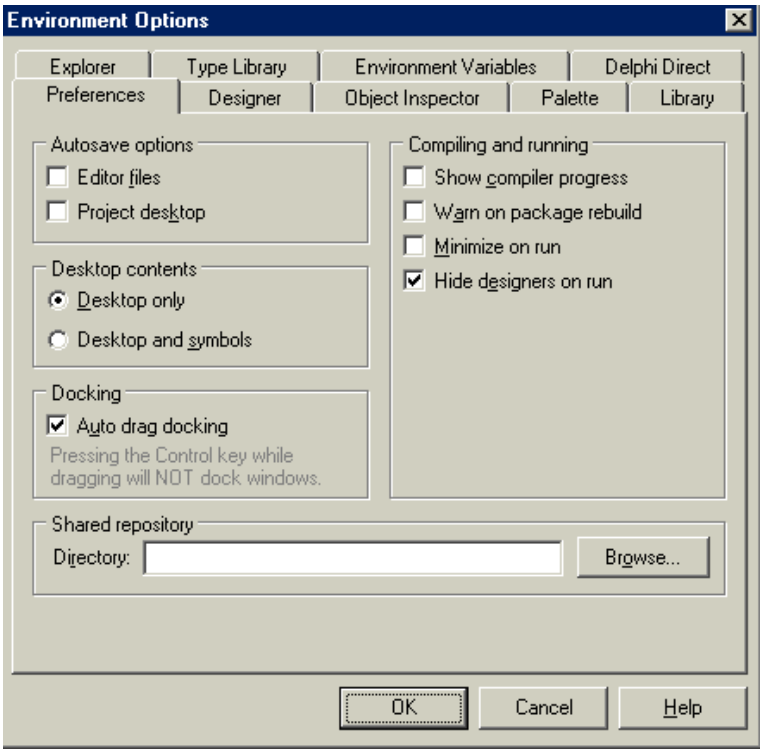

Рис.43. Опция Shared Repository позволяет определить разделяемое хранилище шаблонов.

Разделяемое хранилище

Или, например, вы работаете в сетевом окружении и хотите поделиться своими Формами с другими программистами. Обе эти проблемы позволяет решить концепция разделяемого (Общего) хранилища.

Разделяемое хранилище (Shared Repository) представляет собой папку с двумя файлами, используемыми для хранения информации о расположении шаблонов. Вы можете определить расположение разделяемого хранилища, выполнив следующие действия:

- Выберите команду Tools, Environment Options, и на экран будет

выведено диалоговое окно Environment Options, показанное на рис.43.

- Щелкните на вкладке Preferences вверху диалогового окна (если она не будет сразу выведена на экран).
- В поле ввода Directory введите путь к папке, которую вы хотите использовать в качестве разделяемого хранилища.
- Щелкните по кнопке OK. Теперь при выводе диалогового окна New Items, Delphi будет просматривать эту папку в поисках шаблонов.

После того, как разделяемое хранилище (папка для хранения) определено, вы вольны в выборе места хранения своих шаблонов.

Управление хранилищем объектов

После добавления шаблона в хранилище вы вдруг обнаруживаете, что совсем забыли внести в него последние исправления. Вы можете сделать это, используя диалоговое окно Object Repository, показанное на рис.44. Для его вызова воспользуйтесь командой Tools, Repository. В списке Pages, расположенном в левой части диалогового окна, перечислены страницы, выводимые в диалоговом окне New Items. Управлять этим набором можно с помощью кнопок Add Page (Добавить страницу), Delete Page (Удалить страницу) и Rename Page (Переименовать страницу).

Выбор страницы из списка Pages приводит к заполнению правого списка Objects объектами, содержащимися на этой странице. Если вы выберете элемент [Object Repository], будут показаны все объекты в хранилище. Вы можете перемещать объекты путем их перетаскивания из списка Objects на нужную страницу в списке Pages.

Кнопка Edit Object позволяет вывести диалоговое окно Edit Object Info, показанное на рис.45. С его помощью вы можно редактировать любые Свойства выбранного объекта. Кнопка Delete Object удаляет объект из хранилища, но не удаляет его файлы с диска.

Переключатели, расположенные внизу диалогового окна (рис.44), выполняют две задачи. Первый переключатель New Form, позволяет определить шаблон, используемый при создании новой Формы (File, New, Form). Второй, Main Form, определяет шаблон, используемый для построения главной Формы при создании нового приложения. Для назначения соответствующих шаблонов "по умолчанию" просто выберите объект из списка Objects и отметьте нужную опцию.

Если вы выбираете проект, а не Форму, в диалоговом окне (рис.44) появится новый переключатель - New Project. Он позволяет определить шаблон проекта, используемый при создании нового при-

ложения.

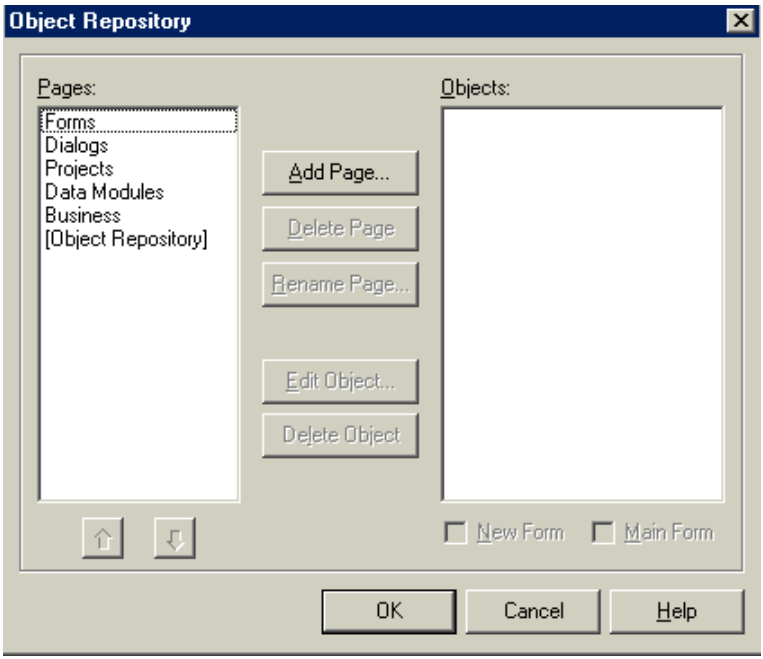

Рис.44. Использование диалогового окна Object Repository для работы с шаблонами.

Наследование Форм

Наследование Форм воплощает лучшие возможности повторного использования шаблонов, которые задаются режимами Use и Copy (см. рис.41). При копировании (Copy), вы создаете дубликат, копию Формы и добавляете в него необходимые компоненты и сам код программы. Неудобство этого метода состоит в том, что изменения не вносятся в шаблон. При использовании режима Use (Использовать) изменения вносятся не только в этот шаблон, но и во все объекты в других проектах, при создании которых был выбран этот режим.

Наследование позволяет создать множество экземпляров шаблона, которые могут отличаться один от другого, как при использовании Copy. Оно так же автоматически вносит изменения в объект хранилища, как при использовании Use.

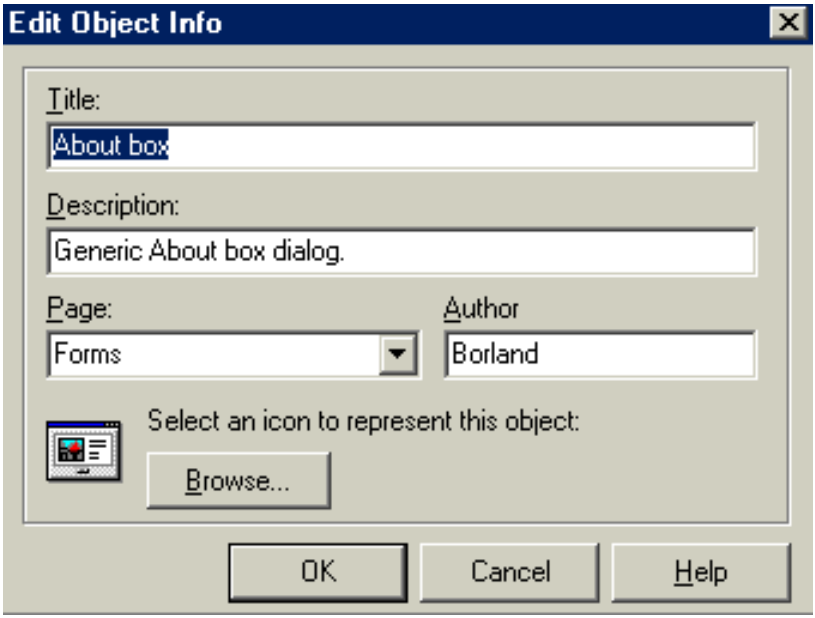

Рис.45. Редактировать Свойства объекта можно в хранилище с помощью диалогового окна Edit Object Info.

Использование наследования

В хранилище объектов содержится несколько примеров наследования Форм. Для того чтобы наследовать одну Форму из другой, выполните следующие действия.

1. Выберите команду File, New, Application. При этом появится пустое приложение.
2. Выберите команду File, New, Other - будет выведено диалоговое окно New Items.
3. Щелкните на вкладке Dialogs - будет выведена страница диалогов.
4. Выберите диалог с пиктограммой Dialog with Help, кнопки которого выровнены вертикально (Vertical) по правой стороне Формы.
5. Выберите опцию Inherit (Наследование) и щелкните на кнопке OK.
6. Система Delphi выведет на экран новую диалоговую Форму.

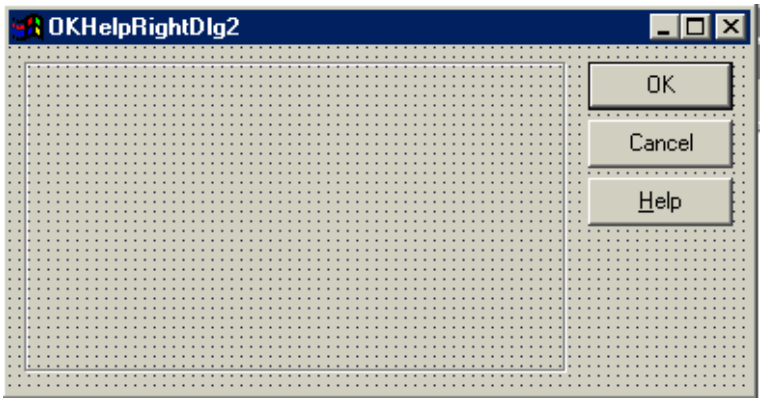

Рис.46. Создание Формы наследника.

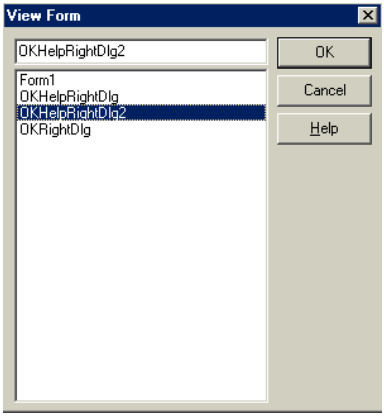

Рис.47. Окно просмотра Форм проекта.

Заголовком нового диалогового окна будет имя OKHelpRightDlg2, как показано на рис.46. Почему Delphi создает это диалоговое окно именно с именем OKHelpRightDlg? Ответ заключается в наследовании (Inherit) Форм. В списке Форм вашего проекта (окно Project Manager из Главного меню View или окно View Form - Главное меню View, Forms, показанное на рис.47) содержится четыре объекта - Form1, OKHelpRightDlg, OKHelpRightDlg2 и OKRightDlg. Поскольку вы наследуете новую Форму из OKHelpRightDlg, Delphi включает его в ваш проект для компиляции вашей программы. Это и приводит к имени новой Формы, как OKHelpRightDlg2. В свою очередь, OKHelpRightDlg - наследник OKRightDlg, а потому последний также включен в проект.

Две родительские Формы, OKHelpRightDlg и OKRightDlg, связываются с проектом так же, как при использовании опции Use, поэтому, решив их изменить, вы измените объекты, хранящиеся в Object Repository.

Цепочка наследования отражена в автоматически генерируемом коде. Описание класса OKHelpRightDlg2 выглядит так:

```
TOKHelpRightDlg2 = class(TOKHelpRightDlg)
private
{Закрытые объявления}
public
{Открытые объявления}
end;
```

Определение класса TOKHelpRightDIg несколько интереснее:

```
TOKHelpRightDIg = class (TOKRightDlg)
HelpBtn: TButton;
procedure HelpBtnClick(Sender: TObject);
private
{Закрытые объявления.}
public
{Открытые объявления.}
end;
```

Как вы можете видеть, OKHelpRightDlg получает компоненты и код от класс OKRightDlg и добавляет объект HelpBtn типа TButton, а также новый обработчик События HelpBtnClick.

Преимущества наследования

Как уже упоминалось ранее, преимущества наследования Форм заключаются в возможности добавления новых компонент и нового кода в создаваемый, наследуемый объект (потомок), и наличие связи с оригиналом шаблона (родителем) в хранилище. Если вы изменяете шаблон, наследуемый объект будет автоматически обновлен. Выполните теперь следующие действия:

1. Выберите команду File, New, Application, и появится пустое приложение.
2. Закройте главную Форму, окно которой озаглавлено Form1.
3. Выберите команду File, New, Other и будет выведено диалоговое окно New Items.
4. Щелкните на вкладке Dialogs, и будет выведена страница диалогов.
5. Выберите стандартный (Standard) диалог с кнопками, выровненными вертикально по правой стороне.
6. Выберите опцию Inherit.

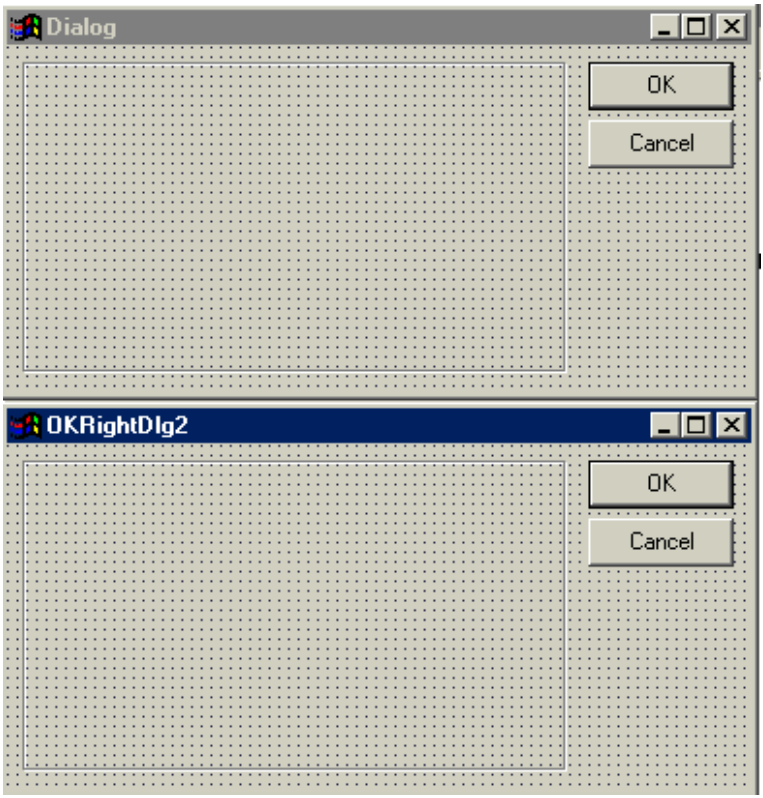

Рис.48 Изменение распространяются только в одном направлении - от родителей к потомкам.

7. Щелкните на кнопке OK, и Delphi выведет новую диалоговую Форму OKRightDlg2.

8. Выберите пункт Главного меню View, Forms, и будет выведено диалоговое окно View Form.

9. Выберите шаблонную Форму OKRightDlg, щелкните на кнопке OK - Delphi выведет новую шаблонную Форму, озаглавленную Dialog и расположенную точно поверх Формы OKRightDlg2.

Обратите внимание, что Форма OKRightDlg2 закрылась (чтобы в этом убедиться, можно подвинуть Форму Dialog). Это связано с тем, что Delphi обновляет Свойства Left и Top одновременно с изменением соответствующих родительских Свойств. Однако в обратном направ-

лении изменения не передаются. Чтобы убедиться в этом, выполните следующие действия:

- Выберите команду View, Forms - появится диалоговое окно View Form.
- Выберите OKRightDlg2 и щелкните мышкой по кнопке OK.
- Переместите OKRightDlg2 в нижнюю часть экрана, а в верхней части у вас останется Форма Dialog.

Ваш экран должен выглядеть примерно так, как показано на рис.48. При перемещении диалога - наследника (OKRightDlg2) на экране, диалог родитель (Dialog) будет оставаться на месте, что иллюстрирует передачу Свойств только в одном направлении.

СОЗДАНИЕ SDI ПРИЛОЖЕНИЙ

Термин SDI (Single Document Interface) дословно означает одно - документный интерфейс и описывает приложения, способные загрузить и использовать одновременно только один документ. Программа Notepad (Блокнот), приведенная на рис.49, является ярким представителем такого класса программ.

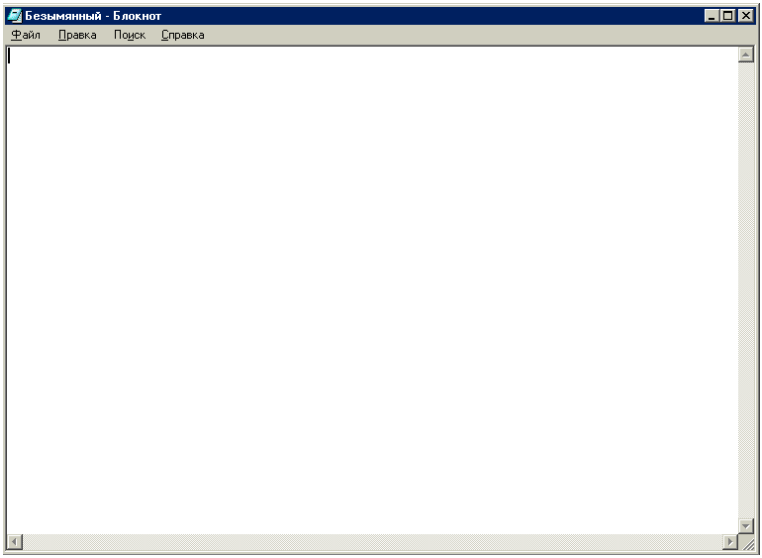

Рис.49. Программа Notepad, как пример
SDI - приложения.

Способность одновременно работать только с одним объектом не мешает приложению использовать дополнительные Формы, например диалоговые окна, панели инструментов и т.д. Примером такой программы является редактор WordPad, показанный на рис.5. Для реализации этих возможностей в Delphi, просто добавьте новую Форму в ваше приложение и установите ее Свойство FormStyle равным fsSizeToolWin или fsToolWindow.

Еще одним примером может служить сама система Delphi - огромное количество панелей инструментов, меню, разнообразных библиотек компонентов, взаимодействующих между собой Форм т.п. Но в целом, она остается SDI - приложением, так как может загрузить и ис-

пользовать одновременно только один объект. Для демонстрации SDI приложения создадим простую программу просмотра изображений.

Построение интерфейса

Обычно первым шагом построения программы является создание интерфейса. Не будем отступать от традиций, и выполним следующие действия:

1. Выберите команду File, New, Application - появится пустое новое приложение. Система Delphi "по умолчанию" создает именно SDI - приложение. Однако хранилище объектов предоставляет возможность назначить и другой шаблон проекта "по умолчанию".
2. Установите следующие Свойства Формы.

Свойство	***Значение***
Caption	Image Viewer
Name	FrmMain
ShowHint	True

3. Поместите на Форму компонент TPanel и установите его Свойства.

Свойство	***Значение***
Caption	-
Align	alTop

4. Поместите три компонента TSpeedButton в TPanel и назовите их spbtnLoad, spbtnStretch и spbtnCenter. Установите следующие их Свойства.

Свойство	***Значение***
spbtnLoad.Hint	Load
spbtnLoad.Left	8
spbtnLoad.Top	8
spbtnStretch.AllowAllUp	True
spbtnStretch.GroupIndex	1
spbtnStretch.Hint	Stretch
spbtnStretch.Left	48
spbtnStretch.Top	8
spbtnCenter.AllowAllUp	True

spbtnCenter.GroupIndex	2
spbtnCenter.Hint	Center
spbtnCenter.Left	80
spbtnCenter.Top	8

5. Поместите еще одну TPanel на Форму и установите следующие Свойства.

Свойство	***Значение***
Align	alClient
Caption	-

6. Поместите компонент TImage во вновь созданную TPanel и установите следующие его Свойства.

Свойство	***Значение***
Align	alClient
Name	imgMain

7. Добавьте на Форму компонент TOpenDialog со следующими Свойствами.

Свойство	***Значение***
Filter	Bitmaps (*.bmp)|*.bmp
Name	opndlgLoad
Options	[ofPathMustExist,ofFileMustExist]

8. Delphi предоставляет вам множество значков для компонента TSpeedButton; они находятся в каталоге IMAGES\BUTTONS. Для нас вполне подойдут следующие установки Свойств Glyph.

Свойство	***Значение***
spbtnLoad.Glyph	FLDROPEN.BMP
spbtnStretch.Glyph	FONTSIZE.BMP
spbtnCenter.Glyph	PICTURE.BMP

Теперь самое время сохранить проект, выбрав в меню команду File, Save Project As. Сохраните Unit1 (Save As), как Main, а проект - как EgSDIApp.

Написание кода

Теперь, после создания интерфейса, перейдем к написанию исходного текста нашего приложения. Сначала загрузите изображение следующим образом:

- Дважды щелкните на компоненте spbtnLoad, Delphi выведет окно редактора, и автоматически создаст обработчик События OnClick.
- Введите код:

```
If opndlgLoad.Execute then
imgMain.Picture.LoadFromFile(opndlgLoad.FileName);
```

Метод opndlgLoad.Execute вызывает стандартное диалоговое окно Windows для открытия файла. Если вы выбрали файл и щелкнули по кнопке OK, этот Метод возвращает True и загружает в Свойство FileName полный путь к имени файла. При щелчке на Cancel или нажатии клавиши <Esc> Метод вернет False.

Компонент TImage предоставляет Свойство Picture, которое является экземпляром класса TPicture. Этот класс обеспечивает работу с растровыми изображениями, пиктограммами и метафайлами. Один из его Методов, LoadFromFile, служит для загрузки изображения по имени файла. Выберите команду Run, Run для компиляции и запуска приложения и попытайтесь открыть картинку. Рисунок должен загрузиться в окно объекта Image.

Закройте приложение и добавьте возможность растягивания изображения:

- Дважды щелкните на компоненте spbtnStretch - Delphi выведет окно редактора, и автоматически создаст обработчик События OnClick.
- Введите код:

```
imgMain.Stretch:= spbtnStretch.Down;
```

Компонент TSpeedButton имеет Свойство Down, которое равно True при нажатой кнопке, а Свойство Stretch класса TImage позволяет растянуть картинку. Для выравнивания картинки по центру воспользуйтесь приведенной выше инструкцией для компонента spbtnCenter и введите следующий код:

```
imgMain.Center:= spbtnCenter.Down;
```

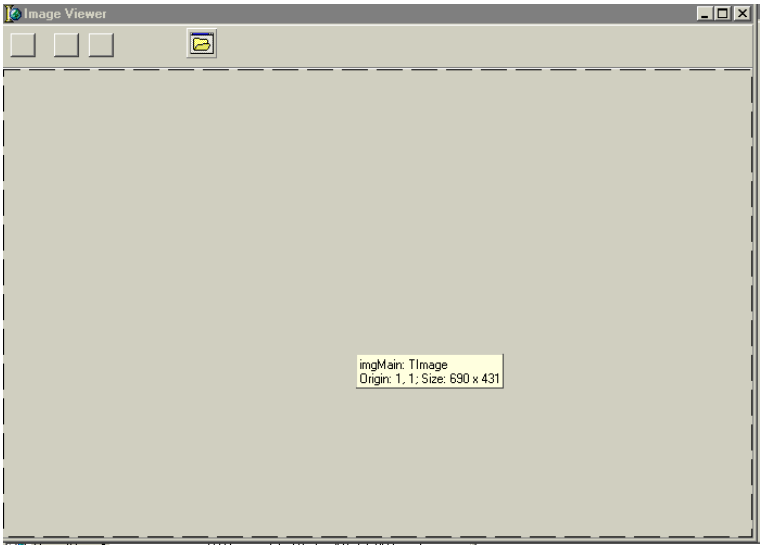

Рис.50. Вид готового приложения Image Viewer.

Теперь приложение полностью готово к работе, а его вид приведен на рис.50. Кнопки Stretch и Center включаются первым щелчком и выключаются при повторном щелчке мышкой.

Использование шаблонов

Возможности Object Repository не ограничиваются хранением Форм, модулей и диалоговых окон - даже целые проекты могут быть сохранены в виде шаблонов. Шаблон проекта может содержать Формы, модули, пользовательский код и использоваться в качестве отправной точки для создания вашего нового проекта.

Система Delphi поставляется с тремя шаблонами проектов:

1. MDI Application - создает полностью функциональный MDI - проект. Родительская Форма включает меню, кнопки SpeedButton и строку состояния. Проект также содержит замещаемый код, реализующий функции меню и управляющий сообщениями в строке состояния.

2. SDI Application - содержит простой SDI - проект. Основная Форма содержит меню, кнопки SpeedButton и строку состояния. В проект также включено диалоговое окно About и реализующий его код.

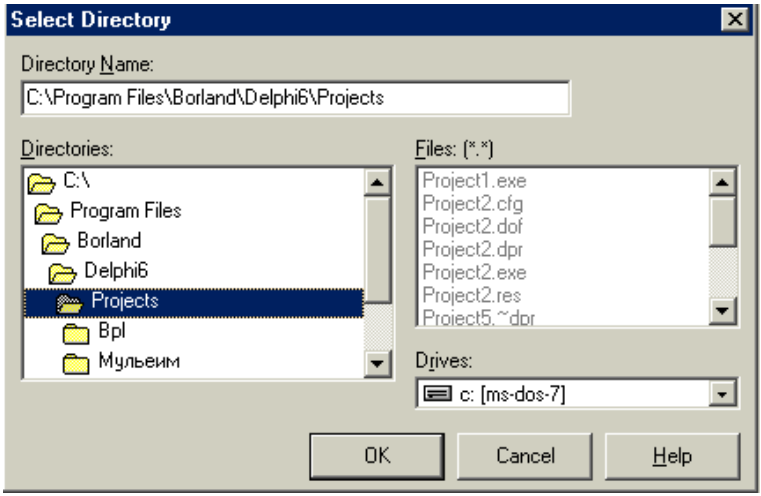

Рис.51. Диалоговое окно Select Directory.

3. Win95 Logo Application - создает проект, моделирующий основные принципы, которые установлены компанией Microsoft для Win95 Logo certification.

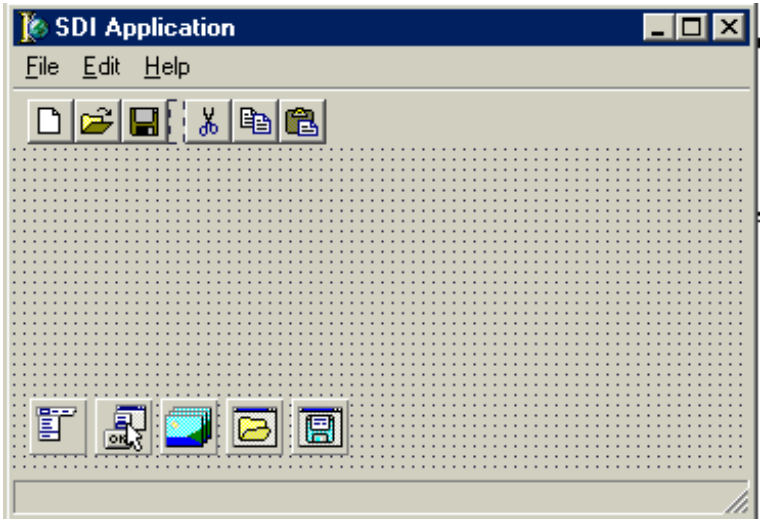

Рис.52. Вид созданного приложения SDI.

Для создания нового SDI - приложения с использованием шаблона выполните следующие действия:

1. Выберите команду File, New, Other - появится диалоговое окно New Items (рис.41).
2. Щелкните на вкладке Projects, что приведет к появлению соответствующей страницы.
3. Выберите пиктограмму SDI Application и щелкните по кнопке OK - появится диалоговое окно Select Directory, показанное на рис.51.
4. Определите каталог для нового проекта. Если вы выберете несуществующий каталог, то система Delphi создаст его.
5. Щелкните на кнопке OK, и новый проект будет создан, а его общий вид приведен на рис.52.

При создании проекта доступна только опция Copy (рис.41) - все файлы проекта копируются в ваш каталог и изменения в них не приведут к изменению шаблона проекта в хранилище.

СОЗДАНИЕ MDI ПРИЛОЖЕНИЯ

Термин MDI (Multiple Document Interface) дословно означает многодокументный интерфейс и описывает приложения, способные одновременно загрузить и использовать несколько документов или объектов. Примером такого приложения может служить диспетчер файлов (File Manager) или текстовой редактор Word.

Обычно MDI - приложения состоят минимум из двух Форм - родительской и дочерней. Свойство FormStyle родительской Формы установлено равным fsMDIForm, а для дочерней Формы нужно установить стиль fsMDIChild.

Родительская Форма служит контейнером, содержащим дочерние Формы, которые заключены в клиентскую область и могут перемещаться, изменять размеры, минимизироваться или максимизироваться. В реальном приложении могут быть дочерние Формы разных типов, например, одна - для обработки изображений, а другая - для работы с текстом.

Создание Форм

В MDI - приложениях, как правило, требуется выводить несколько экземпляров класса Формы. Поскольку каждая Форма представляет собой объект, она должна быть создана перед использованием и освобождена, когда в ней больше не нуждаются. Delphi может это сделать автоматически, а может предоставить такую работу вам.

"По умолчанию", при запуске приложения (обычно SDI), Delphi автоматически создает в проекте по одному экземпляру каждого класса Формы и освобождает их при завершении работы программы. Автоматическое создание обрабатывается, генерируемым в Delphi кодом, в трех местах. Первое - раздел интерфейса в файле модуля Формы:

```
type
TForm1 = class (TForm)
private
{Закрытые объявления.}
public
{Открытые объявления.}
end;
```

В данном фрагменте кода объявляется класс TForm1. Вторым является место, в котором описывается переменная класса:

```
var
Form1: TForm1;
```

Здесь описана переменная Form1, указывающая на экземпляр класса TForm1 и доступная из любого модуля. Обычно она используется во время работы программы для управления Формой.

Третье место находится в исходном тексте проекта, доступ к которому можно получить с помощью меню Project, View Source. Этот код выглядит как:

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

Процесс удаления Форм обрабатывается с помощью концепции владельцев объектов - когда объект уничтожается, автоматически уничтожаются все объекты, которыми он владеет.

Созданная, описанным образом Форма, принадлежит объекту Application и уничтожается при закрытии приложения. Хотя автоматическое создание Форм полезно при разработке SDI - приложений, при создании MDI - приложения оно, как правило, неприемлемо.

В MDI - приложениях, для создания нового экземпляра Формы, нужно использовать конструктор Create класса Форм. Приведенный ниже код создает новый экземпляр TForm1 во время работы программы и устанавливает его Свойство Caption равным 'New Form':

```
Form1:= TForm1.Create(Application);
Form1.Caption:= 'New Form';
```

Обычно в качестве владельца Формы выступает Application, что позволяет автоматически закрывать все Формы по окончании работы приложения. Вы можете также передать параметр Nil, создав Форму без владельца (или владеющую сама собой), но тогда закрывать и уничтожать ее вам придется самостоятельно. В случае возникновения необрабатываемой ошибки такая Форма останется в памяти, что не говорит о высоком профессионализме программиста.

В приведенном выше коде, Form1 указывает только на последнюю созданную Форму. Если вам это не нравится, воспользуйтесь приведенным ниже кодом - возможно, он более точно отвечает вашим запросам:

```
with TForm1.Create(Application) do
Caption:= 'New Form';
```

При разработке MDI - приложения, Метод Show не нужен, так как Delphi автоматически показывает все вновь созданные дочерние MDI - Формы. А вот в случае SDI - приложения, вы обязаны использовать Метод Show.

Даже при динамическом создании Форм, Delphi попытается навязать вам свои услуги по созданию экземпляра каждой Формы. Чтобы отказаться от них, воспользуйтесь диалоговым окном Project Options (Главное меню Project пункт Options), изображенным на рис.53, и удалите все классы Форм из списка Auto - create forms.

Если вы захотите получить доступ к отдельному дочернему экземпляру класса, используйте Свойство MDIChildren, описываемое в следующем разделе.

MDI - Свойства TForm

Объект TForm имеет несколько Свойств, специфичных именно для MDI - приложений. Рассмотрим их более подробно:

ActiveMDIChild

Это Свойство возвращает дочерний объект TForm, имеющий в текущее время фокус ввода. Оно полезно, когда родительская Форма содержит панель инструментов или меню, команды которых распространяются на открытую дочернюю Форму.

Например, представим, что проект использует дочернюю Форму, содержащую элемент TMemo, названный memDailyNotes. Имя класса этой дочерней Формы - TfrmMDIChild. Родительская Форма на своей панели инструментов содержит кнопку Clear (spbtnClear), которая удаляет содержимое memDailyNotes в активной дочерней Форме. Вот как это реализуется в программе:

```
procedure TfrmMDIParent.spbtnClearClick (Sender: TObject);
begin
If not (ActiveMDIChild = Nil) then
If ActiveMDIChild is TfrmMDIChild then
TfrmMDIChild(ActiveMDIChild).memDailyNotes.Clear;
end;
```

В первой строке проверяется, равен ли ActiveMDIChild значению Nil, так как в этом случае обращение к объекту вызовет исключительную ситуацию (не обрабатываемую Delphi самостоятельно).

ActiveMDIChild будет равен Nil, если нет открытых дочерних Форм или Свойство FormStyle дочерней формы не равно fsMDIForm.

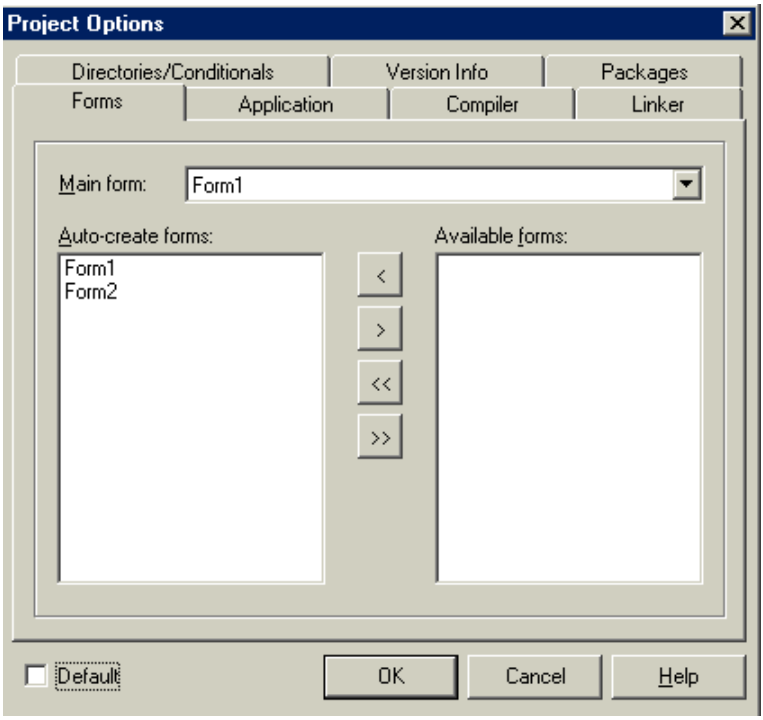

Рис.53. Диалоговое окно Project Options позволяет установить опции для текущего проекта.

Поскольку ActiveMDIChild возвращает объект TForm, компилятор не имеет доступа к memDailyNotes - объекту TfrmMDIChild. Вторая строка проверят соответствие типов, т.е. действительно ли ActiveMDIChild указывает на объект TfrmMDIChild. Третья строка выполняет преобразование типа и вызывает Метод Clear компонента memDailyNotes.

MDIChildren
MDIChildCount

Свойство MDIChildren является массивом объектов TForm, пре-

доставляющих доступ к созданным дочерним Формам. MDIChildCount возвращает количество элементов в массиве MDIChildren. Обычно это Свойство используется при выполнении, какого - либо действия над всеми открытыми дочерними Формами. Вот код сворачивания всех дочерних Форм командой Minimize All:

```
procedure TForm1.mnuMinimizeAllClick(Sender: TObject);
var
iCount: Integers;
begin
for iCount:= MDIChildCount - 1 downto 0 do
MDIChildren[iCount].WindowState:= wsMinimized;
end;
```

Если вы будете сворачивать окна в порядке возрастания элементов массива, цикл будет работать некорректно, так как после сворачивания каждого окна, массив MDIChildren обновляется и пересортировывается, и вы можете пропустить некоторые элементы.

TileMode

Это Свойство перечислимого типа, определяющее, как родительская Форма размещает дочерние Формы, при вызове Метода Tile. В нем используются значения tbHorizontal ("по умолчанию") и tbVertical для размещения Форм по горизонтали и вертикали.

WindowMenu

Профессиональные MDI - приложения позволяют активизировать необходимое дочернее окно, выбрав его из списка в меню. Свойство WindowMenu определяет объект TMenuItem, который Delphi будет использовать для вывода списка доступных дочерних Форм. Для вывода списка TmenuItem, в проекте должно быть меню верхнего уровня. Это меню имеет Свойство Caption, равное swindow.

MDI - События TForm

В MDI - приложении Событие OnActivate запускается только при переключении между дочерними Формами. Если фокус ввода передается из не MDI - Формы в MDI - Форму, генерируется Событие OnActivate родительской Формы, хотя ее Свойство Active никогда и не

устанавливается равным True.

Эта странность на самом деле строго логична - ведь, если бы OnActivate генерировался только для дочерних Форм, не было бы никакой возможности узнать о переходе фокуса ввода от другого приложения.

MDI - Методы TForm

Специфичные для MDI - Форм Методы перечислены ниже:

1. ArrangeIcons - выстраивает пиктограммы минимизированных дочерних Форм в нижней части родительской Формы.
2. Cascade - располагает дочерние Формы каскадом, так что видны все их заголовки.
3. Next и Previous - переходит от одной дочерней Формы к другой, как будто вы нажали Ctrl + Tab или Ctrl + Shift + Tab.
4. Tile - выстраивает дочерние Формы так, что они не перекрываются.

Пример MDI - приложения

В этом разделе мы расширим возможности созданной ранее программы просмотра изображений Image Viwer.

Создание интерфейса

Интерфейс MDI - приложения очень похож на интерфейс разработанного ранее SDI - приложения, но каждое изображение будет выводиться в отдельной, а не в главной Форме.

Выполните следующие действия для создания родительской Формы:

1. Выберите команду File, New, Application, и появится пустое приложение.
2. Установите следующие Свойства.

Свойство	***Значение***
Caption	Image Viewer
FormStyle	fsMDIForm
Name	frmMDIParent

ShowHint	True

3. Поместите на Форму компонент TPanel и установите следующие его Свойства.

Свойство	***Значение***
Align	alTop
Caption	-

4. Поместите три компонента TSpeedButton на TPanel и назовите их spbtnLoad, spbtnStretch и spbtnCenter. Установите следующие их Свойства.

Свойство	***Значение***
spbtnLoad.Hint	Load
spbtnLoad.Left	8
spbtnLoad.Top	8
spbtnStretch.AllowAllUp	True
spbtnStretch.Grouplndex	1
spbtnStretch.Hint	Stretch
spbtnStretch.Left	48
spbtnStretch.Top	8
spbtnCenter.AllowAllUp	True
spbtnCenter.Grouplndex	2
spbtnCenter.Hint	Center
spbtnCenter.Left	80
spbtnCenter.Top	8

5. Свойства Glyph установите те же, что и для SDI - приложения.

6. Добавьте на Форму компонент TOpenDialog и установите следующие его Свойства.

Свойство	***Значение***
Filter	Bitmaps (*.bmp)]*.bmp
Name	opndlgLoad
Options	[ofPathMustExist,ofFileMustExist]

Теперь можно создавать новую дочернюю Форму нашего проекта:

1. Выберите из меню File, New, Form - появится пустая Форма.

2. Установите следующие ее Свойства.

Свойство	***Значение***
FormStyle	fsMDIChild
Name	frmMDIChild
Position	poDefaultPosOnly

3. Поместите компонент TImage во вновь созданную Форму и установите его следующие Свойства.

Свойство	***Значение***
Align	alClient
Name	imgMain

4. Удалите дочернюю Форму из списка автоматически создаваемых Форм следующим образом:

- Выберите команду Project, Options - появится диалоговое окно Project Options, показанное на рис.18.
- Выберите frmMDIChild в списке Auto - create forms.
- Щелкните на кнопке ">". Форма frmMDIChild будет перенесена в список Available forms.
- Щелкните на кнопке ОК.

Теперь самое время сохранить проект, выбрав команду File, Save Project As. Сохраните модуль Unit1 Формы frmMDIParent, как MDIParent, сам проект - как EgMDIApp, а модуль Unit2 Формы frmMDIChild, как MDIChild.

Написание кода

Создав и сохранив интерфейс проекта, перейдем к написанию исходного текста приложения, который будет очень похож на код для SDI - программы. Введите в обработчик События OnClick компонента spbtnLoad следующий код:

```
procedure TfrmMDIParent.spbtnLoadClick (Sender: TObject);
begin
if opndlgLoad.Execute then
with TfrmMDIChild.Create(Application) do
begin
```

```
Caption:= opndlgLoad.FileName;
imgMain.Picture.LoadFromFile(opndlgLoad.FileName);
ClientWidth:= imgMain.Picture.Width;
ClientHeight:= imgMain.Picture.Height;
end;
end;
```

После запуска диалогового окна проекта, создается новый экземпляр дочерней Формы, и именно в нее будет загружаться выбранный вами файл изображения. После загрузки, размеры дочерней Формы изменяются так, чтобы можно было видеть все изображение.

Поскольку модуль MDIParent ссылается на тип TfrmMDIChild, находящийся в модуле MDIChild, после строки implementation следует добавить еще одну строку:

```
uses MDIChild;
```

Теперь можно приступить к компиляции и запуску приложения. Однако заметьте, что, когда вы щелкаете на кнопке Close (кнопка с крестиком справа вверху окна), дочерняя Форма не закрывается, а сворачивается в пиктограмму. Чтобы заставить ее закрыться, следует добавить в код обработчика События OnClose Формы TfrmMDIChild маленькую деталь - изменить Свойство Action:

```
Action:= caFree;
```

Компоненты TspeedButton - Stretch и Center выполняют те же функции, что и в SDI - приложении, однако их обработчики События OnClick следует изменить следующим образом:

```
If not (ActiveMDIChild = Nil) then
If ActiveMDIChild is TfrmMDIChild then
TfrmMDIChild(ActiveMDIChild).imgMain.Stretch:=
spbtnStretch.Down;
```

и

```
If not (ActiveMDIChild = Nil) then
If ActiveMDIChild is TfrmMDIChild then
TfrmMDIChild(ActiveMDIChild).imgMain.Center:=
spbtnCenter.Down;
```

Остается последняя проблема - состояния кнопок Stretch и Center одинаковы для всех дочерних Форм. Для решения этой задачи добавьте в обработчик События OnActivate Формы TfrmMDIChild строки:

```
frmMDIParent.spbtnStretch.Down:= imgMain.Stretch;
frmMDIParent.spbtnCenter.Down:= imgMain.Center;
```

И, наконец, самый последний штрих - в модуле MDIChild добавьте после строки implementation строку.

```
uses MDIParent;
```

Компилируйте, запускайте и смотрите свое MDI - приложение - оно полностью готово к работе, а его внешний вид будет подобен, показанному на рис.50. Каждый загруженный рисунок будет открываться в новом окне, и для каждого окна кнопки Stretch и Center будут работать независимо. При первой загрузке рисунка, его окно нужно Развернуть кнопкой с квадратиком в правом верхнем углу - для этого используйте полосы прокрутки основного родительского окна. Переход между открытыми окнами выполняется клавишами Shift + Ctrl.

РИСОВАНИЕ

Система Delphi позволяет выводит на Форме почти любые графические изображения. Эти возможности достигаются использованием графических компонент системы.

Графические компоненты

В стандартную библиотеку визуальных компонент Delphi входит несколько объектов, с помощью которых можно придать своей программе совершенно оригинальный вид. Это объекты TImage, TShape и TBevel (панель Additional), которые мы теперь рассмотрим более подробно:

- TImage позволяет поместить графическое изображение в любое место на вашей Форме. Объект очень прост в использовании - выберите его на странице Additional и поместите в нужное место Формы. Картинку можно загрузить во время дизайна Формы, используя свойство Picture (Инспектор Объектов). Картинка должна храниться в файле в формате BMP (Bitmap), WMF (Windows Meta File) или ICO (Icon).

Как известно, форматов хранения изображений гораздо больше, например, наиболее известны PCX, GIF, TIFF, JPEG. Для включения в программу изображений в этих форматах нужно либо перевести их в формат BMP, либо найти библиотеки третьих фирм, в которых есть аналог TImage, понимающий данные форматы.

При проектировании следует помнить, что изображение, помещенное на Форму во время дизайна, включается в файл .DPR и затем прикрепляется к EXE файлу программы. Поэтому такой EXE файл может получиться достаточно большой. Как альтернативу, можно рассмотреть загрузку картинки во время выполнения программы - для этого у свойства Picture (которое является объектом со своим набором свойств и методов) есть специальный метод LoadFromFile. Это делается, например, так:

```
If OpenPictureDialog1.Execute then
Image1.Picture.LoadFromFile(OpenPictureDialog1.FileName);
```

Важными являются свойства объекта Center и Stretch, которые имеют булевский тип (т.е. могут принимать значения True - Истина или False - Ложь). Если Center установлено в True, то центр изображения будет совмещаться с центром объекта TImage. Если Stretch уста-

новлено в True, то изображение будет сжиматься или растягиваться таким образом, чтобы заполнить весь объект TImage.

- Компонент TShape это простейшие графические объекты на Форме типа круга, квадрата и т.п. Вид объекта указывается в свойстве Shape. Свойство Pen определяет цвет и вид границы объекта. Свойство Brush задает цвет и вид заполнения объекта. Эти свойства можно менять, как во время дизайна объекта, так и во время выполнения программы.
- TBevel - объект, который используется для украшения программы и может принимать вид рамки или линии. Объект предоставляет меньше возможностей по сравнению с TPanel, но не занимает компьютерных ресурсов. Внешний вид компонента указывается с помощью свойств Shape и Style.

Свойство объектов Canvas

У ряда объектов из библиотеки визуальных компонент есть свойство Canvas (Канва), которое предоставляет простой способ рисования объектов. Это объекты - TBitmap, TComboBox, TDBComboBox, TDBGrid, TDBListBox, TDirectoryListBox, TDrawGrid, TFileListBox, TForm, TImage, TListBox, TOutline, TPaintBox, TPrinter, TStringGrid.

Canvas является объектом, объединяющим в себе поле для рисования, карандаш (Pen), кисть (Brush) и шрифт (Font). Canvas обладает также рядом графических методов: Draw, TextOut, Arc, Rectangle и т.д. Используя Canvas, вы можете воспроизводить на Форме любые графические объекты - картинки, многоугольники, текст и т.п. без использования компонент TImage, TShape и TLabel (т.е. без использования дополнительных ресурсов компьютера). Однако при этом, вы должны обрабатывать событие OnPaint того объекта, на Канве которого вы рисуете.

Свойства Canvas

Рассмотрим подробнее некоторые свойства объекта Canvas:

- Brush - кисть, является объектом со своим набором свойств.
- Bitmap - картинка размером 8x8. Используется для заполнения (заливки) области на экране.
- Color - цвет заливки.
- Style - определенный стиль заливки. Это свойство конкурирует со свойством Bitmap - какое свойство вы определили последним, то

и будет определять вид заливки.

- Handle - данное свойство дает возможность использовать кисть в прямых вызовах процедур Windows.
- ClipRect - прямоугольник, на котором происходит графический вывод.
- CopyMode - свойство определяет, каким образом будет происходить копирование (метод CopyRect) изображения на данную Канву из другого места: один к одному, с инверсией изображения и т.д.
- Font - шрифт, которым выводится текст (метод TextOut).
- Pen - карандаш, который определяет вид линий и также как кисть (Brush) является объектом с набором свойств:

  - Color - цвет линии.
  - Handle - для прямых вызовов процедур Windows.
  - Mode - режим вывода: простая линия, с инвертированием, с выполнением исключающего "или" и т.д.
  - Style - стиль вывода: линия, пунктир и т.п.
  - Width - ширина линии в точках (пикселах).
  - PenPos - текущая позиция карандаша. Карандаш рекомендуется перемещать с помощью метода MoveTo, а не прямой установкой данного свойства.
  - Pixels - двухмерный массив элементов изображения (Pixel). С его помощью вы получаете доступ к каждой отдельной точке изображения.

Методы Canvas

Методы для рисования простейшей графики - это Arc, Chord, LineTo, Pie, Polygon, PolyLine, Rectangle, RoundRect. При прорисовке линий в этих методах используются карандаш (Pen) Канвы, а для заполнения внутренних областей - кисть (Brush).

Методы для вывода картинок на Канву - Draw и StretchDraw. В качестве параметров указываются прямоугольник и графический объект для вывода (это может быть TBitmap, TIcon или TMetafile). Метод StretchDraw отличается тем, что растягивает или сжимает картинку так, чтобы она заполнила весь указанный прямоугольник.

Методы для вывода текста - TextOut и TextRect. При выводе текста используется шрифт (Font) Канвы. При использовании TextRect текст выводится только внутри указанного прямоугольника. Длину и высоту текста можно узнать с помощью функций TextWidth и TextHeight.

Объект TPaintBox

На странице System Палитры Компонент есть объект TPaintBox, который можно использовать для построения приложений типа графического редактора или, например, в качестве места построения графиков.

Никаких ключевых свойств, кроме Canvas объект TPaintBox не имеет. Собственно, этот объект является просто канвой для рисования. Важно помнить, что координаты указателя мыши, передаваемые обработчику соответствующих событий (OnMouseMove и т.д.), являются относительными, т.е. это смещение мыши относительно левого верхнего угла объекта TPaintBox, а не относительно левого верхнего угла Формы.

Пример загрузки рисунка

Для загрузки рисунка в окно типа Image нужно поместить его на Форму, затем поместить на нее кнопку и сам диалог OpenPictureDialog. Назовем, например, кнопку Load (Загрузить) и после двойного щелчка по ней поместим в процедуру следующий код:

```
procedure TForm1.LoadClick(Sender: TObject);
begin
If OpenPictureDialog1.Execute then
Image1.Picture.LoadFromFile(OpenPictureDialog1.FileName);
end;
```

Теперь, при запуске приложения, после щелчка по кнопке будет появляться стандартное окно открытия файла Windows (Открыть или Open), в котором можно выбрать любой графический файл и, щелкнув по нему загрузить в окно Image. Если установить Свойство Stretch окна Image в положение True, то при загрузке, рисунок будет точно вписываться в окно Image.

Код основного модуля такой программы будет выглядеть следующим образом:

```
unit Unit1;
interface
uses Windows, Messages, SysUtils, Variants, Classes,
Graphics, Controls, Forms,
Dialogs, StdCtrls, ExtCtrls, ExtDlgs;
```

```
type
TForm1 = class(TForm)
Image1: TImage;
Load: TButton;
OpenPictureDialog1: TOpenPictureDialog;
procedure LoadClick(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;
var
Form1: TForm1;
implementation
{$R *.dfm}
```

Далее идет сама процедура, приведенная выше и последний оператор END с точкой.

ПЕЧАТЬ

Delphi позволяет реализовать режимы печати в обычном текстовом и графическом режиме работы.

Печать в текстовом режиме

Если вам нужно напечатать на принтере документ в текстовом режиме, то это делается следующим образом - с принтером вы работаете, как с обычным текстовым файлом, за исключением того, что вместо процедуры AssignFile нужно вызывать процедуру AssignPrn. В приведенном ниже примере на принтер выводится одна строка текста:

```
procedure TForm1.Button1Click(Sender: TObject);
Var
To_Prn: TextFile;
begin
AssignPrn(To_Prn);
Rewrite(To_Prn);
Writeln(To_Prn, 'Pinter in Text Mode');
CloseFile(To_Prn);
end;
```

Здесь необходимо пояснить, что по сравнению с Borland Pascal 7.0 в Delphi, в модуле System изменены названия некоторых функций и переменных:

```
AssignFile вместо Assign
CloseFile вместо Close
TextFile вместо Text
```

Иногда в программе требуется просто получить твердую (бумажную) копию экранной Формы. В Delphi это делается очень просто - у объекта TForm есть метод Print, который и нужно вызвать в нужный момент.

Приведем теперь реальный фрагмент кода программы, полностью описывающий печать многостраничного документа на принтере:

```
procedure TForm1.SpeedButton36Click(Sender: TObject);
var
I, Start, Stop: Integer;
```

```
begin
PrintDialog1.Options:= [poPageNums, poSelection];
PrintDialog1.FromPage:= 1;
PrintDialog1.MinPage:= 1;
PrintDialog1.ToPage:= PageControl1.PageCount;
PrintDialog1.MaxPage:= PageControl1.PageCount;
if PrintDialog1.Execute then
  begin
    with PrintDialog1 do
    begin
      if PrintRange = prAllPages then
        begin
        Start:= MinPage - 1;
        Stop:= MaxPage - 1;
      end
      else
      if PrintRange = prSelection then
      begin
        Start:= PageControl1.ActivePage.PageIndex;
        Stop:= Start;
        end
      else  { PrintRange = prPageNums }
      begin
        Start:= FromPage - 1;
        Stop:= ToPage - 1;
      end;
    end;
    with Printer do
    begin
      BeginDoc;
      for I:= Start to Stop do
      begin
        PageControl1.Pages[I].PaintTo(Handle, 10, 10);
        if I <> Stop then
          NewPage;
      end;
      EndDoc;
    end;
  end;
end;
```

Графическая печать

Рассмотрим теперь, как из программы, созданной в Delphi, можно вывести на печать графическую информацию. Для этого есть специальный объект Printer (класса TPrinter). Он становится доступен, если к программе подключить модуль Printers (т.е. добавить имя модуля в разделе uses).

С помощью этого объекта печать на принтере графической информации становится не сложнее вывода этой информации на экран. Основным является то, что Printer предоставляет разработчику свойство Canvas (работа с Канвой описана в предыдущем параграфе) и методы, выводящие содержание Канвы на принтер.

Свойства Printer

Теперь рассмотрим подробнее свойства Delphi - объекта Printer:

- Aborted - тип булевский. Показывает, прервал ли пользователь работу принтера методом Abort.
- Canvas - канва, место для вывода графики.
- Fonts - список доступных шрифтов.
- Handle - используется при прямых вызовах процедур Windows.
- Orientation - ориентация страницы - вертикально или горизонтально.
- PageWidth, PageHeight, PageNumber - ширина, высота и номер страницы соответственно.
- Printers - перечисляет все установленные в системе принтеры.
- PrinterIndex - указывает, какой из них является текущим. Чтобы печатать на принтере "по умолчанию" здесь должно быть значение 1.
- Printing - тип булевский. Показывает, начата ли печать (методом BeginDoc).
- Title - заголовок для Print Manager и перед выводом на сетевом принтере.

Методы Printer

Объект Printer имеет следующие методы:

- Abort - прерывает печать, начатую методом BeginDoc.

- BeginDoc - вызывается перед тем, как начать рисовать на Канве.
- EndDoc - вызывается, когда все необходимое на Канве уже нарисовано. Принтер начинает печатать именно после этого метода.
- NewPage - переход на новую страницу.

Остальными методами объекта Printer в обычных случаях пользоваться не приходится.

Итак, порядок вывода на печать графической информации выглядит следующим образом - на Канве выполняется метод BeginDoc (Canvas), затем рисуем все, что нужно, а при необходимости поместить информацию на нескольких листах, вызываем метод NewPage и посылаем нарисованное на принтер, выполняя метод EndDoc.

МУЛЬТИМЕДИА

Система Delphi позволяет легко и просто включать в программу такие мультимедийные объекты, как звуки, видео и музыку. В этой главе обсуждается, как это сделать, используя встроенный в Delphi компонент TMediaPlayer. Мы подробно рассмотрим управление этим компонентом и получение информации о его текущем состоянии.

Возможности мультимедиа

Дадим наиболее общее определение и скажем, что “мультимедиа” - это термин, относящийся почти ко всем формам анимации, звука и видео, которые используются на компьютере. Давая такое определение, нужно сказать, что в данной главе мы будем иметь дело с подмножеством мультимедиа, которое включает:

1. Показ видео в формате Microsoft Video for Windows (файлы с расширением AVI).
2. Воспроизведение звука и музыки из MIDI и WAVE файлов.

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

Для проигрывания файлов мультимедиа может потребоваться наличие некоторого оборудования и программного обеспечения. Так для воспроизведения звуков нужна звуковая карта и колонки. Для воспроизведения AVI (или WFW) требуется установить программное обеспечение Microsoft Video.

Компонент системы Delphi TMediaPlayer дает нам доступ ко всем основным возможностям программирования мультимедиа и очень прост в использовании. Фактически, он настолько прост, что многим, даже начинающим программистам будет проще создать свою первую программу на Delphi, проигрывающую видео или музыку, нежели показывающую классическую надпись "Hello World", которую обычно используют в примерах программирования на Java.

Простоту использования этого компонента можно воспринимать таким образом:

- С одной стороны - это дает возможность любому программисту

создавать мультимедиа приложения.

- С другой стороны, можно обнаружить, что в компоненте реализованы не все возможности. Если вы захотите использовать низкоуровневые функции, то придется копаться достаточно глубоко, используя язык Delphi.

Здесь мы не описываем подробности внутренних вызовов мультимедийных функций при работе этого компонента. Все что нужно в данный момент знать - это то, что компонент называется TMediaPlayer, и что он дает доступ к набору подпрограмм, созданных Microsoft и называемых Media Control Interface (MCI). Эти подпрограммы дают программисту простой доступ к широкому кругу устройств мультимедиа.

Компонент TMediaPlayer

Для начала давайте создадим новый проект, затем поместим компонент TMediaPlayer (Панель System) на Форму, как показано на рис.54.

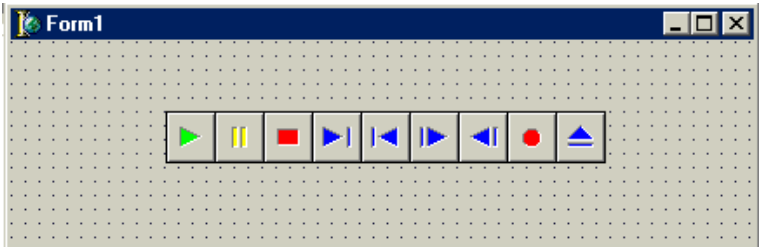

Рис.54. Компонент TMediaPlayer на Форме.

Компонент TMediaPlayer оформлен, как панель управления устройством воспроизведения мультимедиа. Как на магнитофоне, здесь есть кнопки “воспроизведение”, “перемотка”, “запись” и т.д.

Поместив компонент на Форму, вы увидите, что Инспектор Объектов содержит свойство FileName. Щелкните дважды на этом свойстве и в появившемся на экране окне выберите некоторый файл с расширением AVI, WAV или MID. Далее в Инспекторе объектов нужно установить свойство AutoOpen в положение True.

После выполнения этих шагов программа готова к запуску. Запустив программу, нажмите зеленую кнопку Воспроизведения и вы увидите видео ролик (если выбрали AVI) или услышите звук (если выбра-

ли WAV или MID). Видео ролик будет показан в новом окне, которое откроется на экране после начала воспроизведения. Если этого не произошло или появилось сообщение об ошибке, то возможны два варианта:

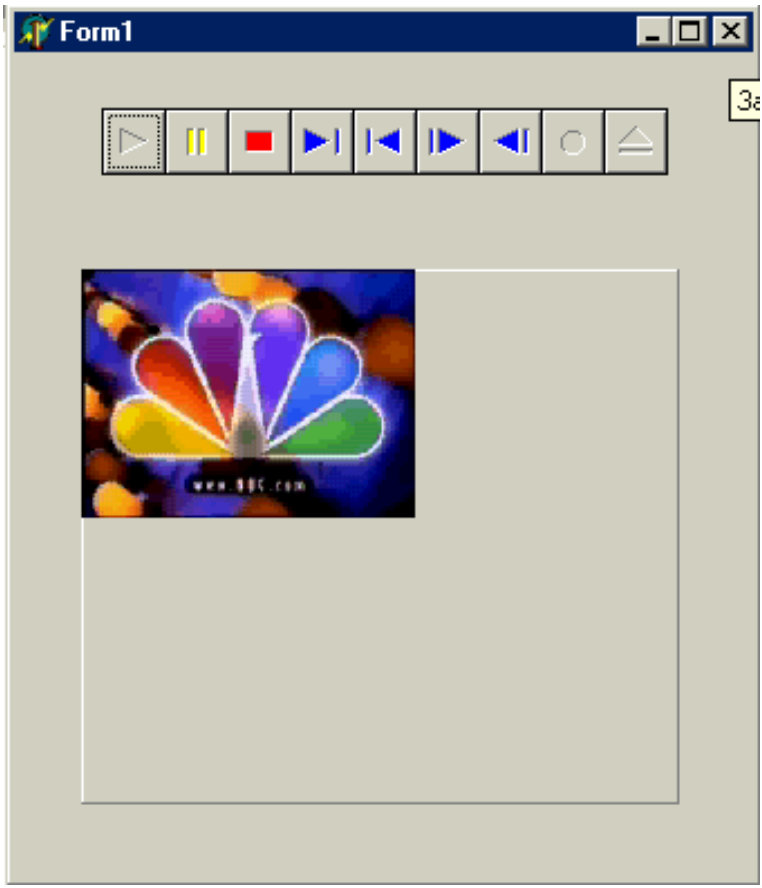

Рис.55. Воспроизведение AVI на панели.

- Вы ввели неправильное имя файла - использовали файл с другим расширением.

- Вы не настроили правильным образом мультимедиа в Windows. Это означает, что у вас нет соответствующего оборудования, либо не установлены нужные драйверы. Установка и настройка драйверов про-

изводится в Control Panel, требования к оборудованию компьютера приводятся в любой книге по мультимедиа (нужна звуковая карта, например, совместимая с Sound Blaster).

Отметим еще одно важное свойство компонента TMediaPlayer - Display. Изначально оно не заполнено и видео воспроизводится в отдельном окне. Однако в качестве экрана для показа ролика можно использовать, например, объект Панель. На Форму нужно поместить компонент TPanel и убрать текст из Свойства Caption в Инспекторе объектов. Далее, для TMediaPlayer, в свойстве Display выбрать из списка Panel1. После этого надо запустить программу и нажать кнопку Воспроизведение (рис.55)

Иногда приходится предоставлять пользователям простой путь для проигрывания максимально широкого круга файлов. Это означает, что вам нужно будет дать пользователю доступ к жесткому диску или CD - ROM, а затем позволить ему выбрать и воспроизвести некоторый файл. В этом случае, на Форме располагается объект TMediaplayer и дополнительные компоненты, предоставляющие возможность выбора файла и управления воспроизведением.

Иногда программист может захотеть скрыть от пользователя существование компонента TMediaPlayer. То есть, воспроизвести звук или видео без того, чтобы пользователь заботился об их источнике. В частности, звук может быть частью презентации.

Например, показ, какого - нибудь графика на экране может сопровождаться объяснением, записанным в WAV файле. В течение презентации пользователь даже не знает о существовании TmediaPlayer - он работает в фоновом режиме. Для этого компонент делается невидимым (Свойство Visible устанавливается в положение False) и управляется программно.

Программы с мультимедиа

Теперь мы рассмотрим пример построения приложения с мультимедиа первого типа. Создайте новый проект (File, New, Aplication). Поместите TMediaPlayer на Форму, поместите компоненты TFileListBox (выбор файлов), TDirectoryListBox (выбор директорий), TDriveComboBox (выбор дисков) и TFilterComboBox для выбора типа файла. В свойстве FileList для DirectoryListBox1 и FilterComboBox1 поставьте FileListBox1. В Свойстве DirList для DriveComboBox1 поставьте DirectoryListBox1.

В Свойстве Filter для FilterComboBox1 укажите требуемые расши-

рения файлов (дважды щелкнув по этому Свойству):

```
AVI File(*.avi)      *.avi
WAVE File(*.wav)     *.wav
MIDI file(*.MID)     *.mid
```

Пусть по двойному щелчку мышкой в окне FileListBox1 сразу будет воспроизводиться выбранный файл. В обработчике события OnDblClick для FileListBox1 укажите

```
procedure TForm1.FileListBox1DblClick(Sender:TObject);
begin
with MediaPlayer1 do
begin
Close;
FileName:=FileListBox1.FileName;
Open;
Play;
end;
end;
```

В обработчике событий по одному щелчку OnClick нужно написать код - это предоставит возможность простого выбора файлов

```
procedure TForm1.FileListBox1Click(Sender:TObject);
begin
with MediaPlayer1 do
begin
Close;
FileName:=FileListBox1.FileName;
Open;
end;
end;
```

Внешний вид такой Формы представлен на рис.56. Сохраните проект, запустите его, выберите нужный файл и дважды щелкните по нему мышкой. MediaPlayer должен воспроизвести этот файл в отдельном окне. При одинарном щелчке по файлу он будет загружен в проект, но воспроизведение включится только после нажатия кнопки Воспроизведение. Отметим, что в Инспекторе объектов для MediaPlayer в поле FileName должен быть указан любой файл для воспроизведения.

Выбор другого файла в окне TFileListBox, заменит значение этого поля именем выбранного файла.

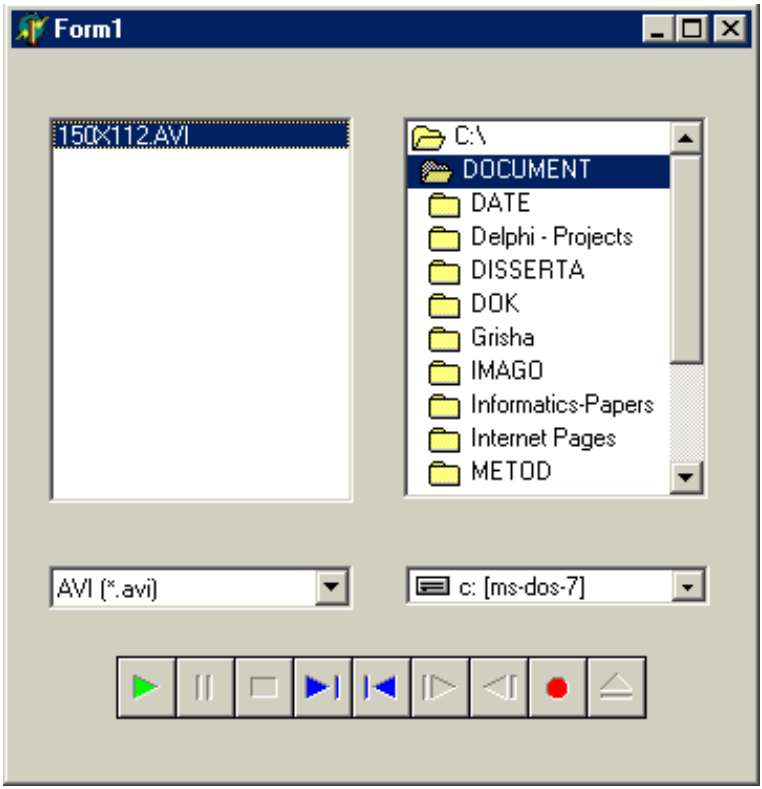

Рис.56. Начальный вид проекта с выбором файла.

Как уже говорилось выше, видео ролик можно воспроизводить внутри Формы, например, на объекте Панель. Модифицируем проект и добавим туда панель TPanel (рис.57). В Свойстве Display для MediaPlayer1 укажите Panel1. Нужно убрать надпись с Панели (Свойство Caption), а Свойство BevelOuter объекта TPanel можно установить в положение bvNone. Чтобы переключаться при воспроизведении с окна на панель, поместите на Форму компонент TCheckBox, и в обработчике события OnClick запишите код:

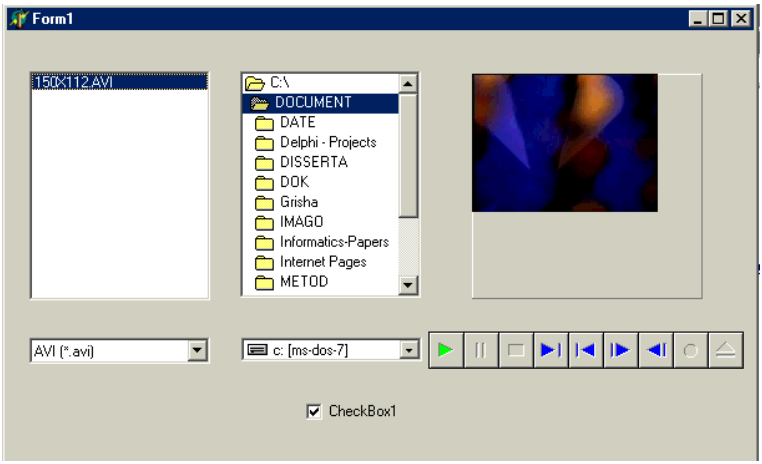

Рис.57. Добавлена панель для видео и переключатель окно/панель.

```
procedure TForm1.CheckBox1Click(Sender: TObject);
var
Start_From: Longint;
begin
With MediaPlayer1 do
begin
If FileName='' then Exit;
Start_From:=Position;
Close;
Panel1.Refresh;
If CheckBox1.Checked then
Display:=Panel1
else
Display:=NIL;
Open;
Position:=Start_From;
Play;
end;
end;
```

При этом в Свойстве Display нужно выключить установку Panel1. Запустите проект и воспроизведите видео ролик из выбранного в TFileListBox файла. Пощелкайте мышкой на CheckBox, изменяя место

воспроизведения этого видео файла - в отдельном окне или на Панели.

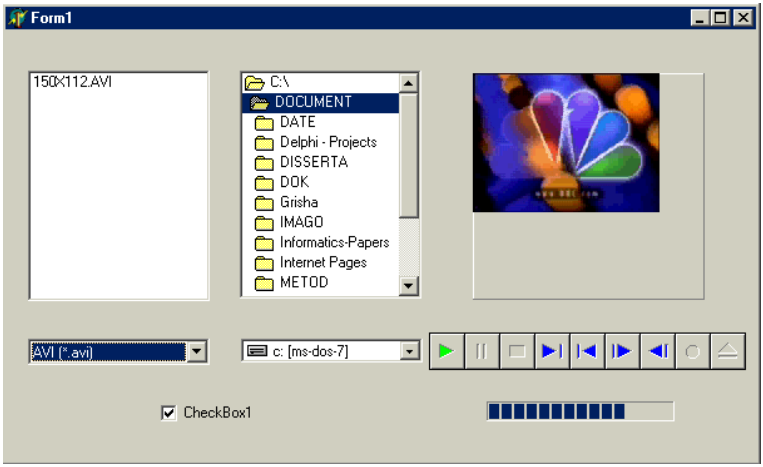

Рис.58. Приложение для воспроизведения AVI, WAV и MDI файлов.

Во время выполнения программы может потребоваться отобразить текущее состояние объекта MediaPlayer и самого видео ролика (время, прошедшее с начала воспроизведения, длину ролика и т.д.). Давайте добавим в проект прогресс - индикатор (TProgressBar - Панель Win32), который отобразит в процентах, сколько прошло времени (рис.58). Для обновления показаний индикатора можно воспользоваться таймером. Поместите на Форму объект TTimer, установите для него Interval = 100 (100 миллисекунд). В обработчике события OnTimer нужно записать:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
with MediaPlayer1 do
If FileName<>'' then
ProgressBar1.Position:=Round(100*Position/Length);
end;
```

Запустите проект, выберите видео файл (AVI) и щелкните на нем два раза мышкой. При воспроизведении ролика прогресс - индикатор будет отображать процент, соответствующий прошедшему времени.

СВОЙСТВА DELPHI

В этой и следующих главах мы будем рассматривать способы изменения Свойств, Методов и Событий для различных компонент Delphi во время выполнения, работы программы. Такие действия не намного сложнее, чем изменение Свойств в режиме проектирования, разработки проекта с помощью Object Inspector. Далее вы увидите, что язык Object Pascal, лежащий в основе Delphi, дает вам полное управление работой приложения. Кроме того, мы рассмотрим понятие информации периода выполнения и научимся использовать ее в целях создания гибких и универсальных приложений.

Каждый компонент, который вы помещаете на Форму, имеет свое отражение в окне Инспектора Объектов (Object Inspector). Окно Object Inspector имеет две "странички" - Properties (Свойства) и Events (События). Создание программы в Delphi сводится к "нанесению" компонент на Форму (которая, кстати, также является компонентом) и настройке взаимодействия между ними с помощью:

- Изменения значения Свойств этих компонент.
- Задания определенных реакций на некоторые События.

Более подробно События мы рассмотрим в следующей главе, а сейчас остановимся на Свойствах, но в меру необходимости, будем затрагивать и создание откликов на определенные События.

Свойства компонент

Свойство является важным атрибутом каждого компонента или объекта Delphi. Для пользователя (программиста) Свойство выглядит, как простое поле, какой - либо структуры, содержащее некоторое значение. Однако, в отличие от "просто" поля, любое изменение значения некоторого Свойства любого компонента сразу же приводит к изменению визуального представления этого компонента, поскольку Свойство включает в себя Методы (Действия), связанные с чтением и записью этого поля. Свойства служат двум главным целям. Во - первых, они определяют внешний вид Формы или компонента. А во - вторых, Свойства определяют само поведение Формы или компонента в тех или иных ситуациях. Существует несколько типов Свойств, в зависимости от их "природы", т.е. внутреннего устройства:

- Простые Свойства - это Свойства, значения которых являются

числами или строками. Например, Свойства Left (Слева) и Top (Сверху) принимают целые значения (числа), определяющие положение левого верхнего угла компонента или Формы. Свойства Caption (Заголовок) и Name (Имя) представляют собой строки и определяют заголовок и имя компонента или Формы.

- Перечислимые Свойства - эти Свойства могут принимать некоторые значения из предопределенного набора (списка) возможных значений. Простейший пример - Свойство типа Boolean, которое может принимать значения True (Истина) или False (Ложь).
- Вложенные Свойства - поддерживают вложенные значения или объекты, а Object Inspector изображает знак "+" слева от названия таких Свойств. Имеется два вида вложенных Свойств:

- Множества.
- Комбинированные значения.

Object Inspector изображает множества в квадратных скобках. Если множество пустое, оно отображается, как пустые скобки [ ]. Установки для вложенных Свойств типа "множество" обычно имеют значения типа Boolean. Наиболее распространенным примером такого Свойства является Свойство Style с вложенным множеством булевых значений (рис.59).

Комбинированные значения отображаются в Инспекторе Объектов, как коллекция некоторых величин, каждая из которых имеет свой тип данных (рис.59). Некоторые Свойства, например, Font, для изменения своих значений имеют возможность вызвать диалоговое окно. Для этого достаточно щелкнуть мышкой маленькую кнопку с тремя точками в правой части строки Инспектора Объектов, показывающей данное Свойство.

Delphi позволяет легко манипулировать Свойствами компонент, как в режиме проектирования (Design Time), так и в режиме выполнения программы (Run Time). В режиме проектирования манипулирование Свойствами осуществляется с помощью Дизайнера Форм (Forms Designer) или на страничке Properties Инспектора Объектов (рис.59), а Событиями на вкладке Events (рис.60).

Например, для того чтобы изменить Свойства Height (Высота) и Width (Ширина) кнопки Button (рис.61), достаточно "зацепить" мышкой за любой ее угол и раздвинуть до нужного представления (размера). Того же результата можно добиться, просто подставив новые значения Свойств Height и Width в окне Object Inspector (рис.59).

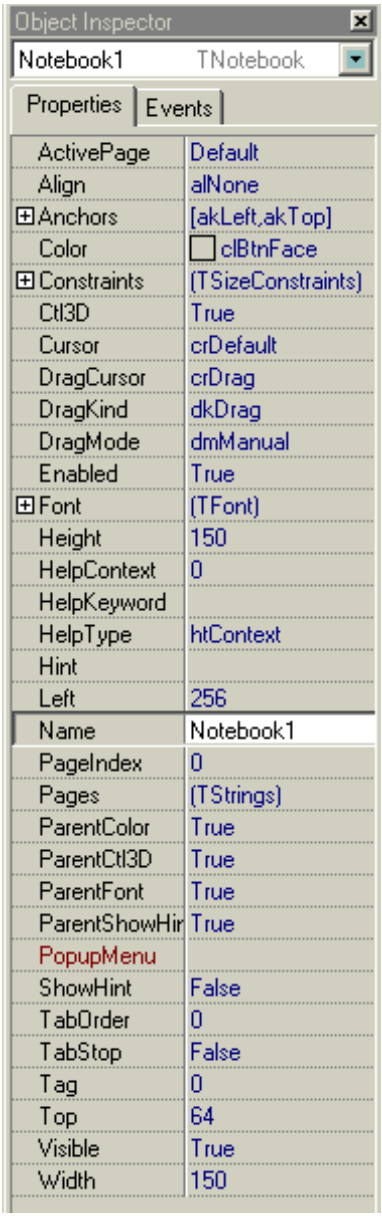

Рис.59. Инспектор объектов - вкладка Свойства.

С другой стороны, в режиме выполнения программы пользователь (программист) имеет возможность не только манипулировать всеми Свойствами, отображаемыми в Инспекторе Объектов, но и управлять более обширным их списком. В следующем параграфе мы рассмотрим, как это делается.

Управление Свойствами

Все изменения значений Свойств компонент в режиме выполнения программы должны осуществляться путем прямой записи строк кода на языке Паскаль. В режиме выполнения невозможно использовать Object Inspector. Однако доступ к Свойствам компонент довольно легко получить программным путем. Все, что вы должны сделать для изменения какого - либо Свойства это написать простую строчку кода аналогично следующей:

```
MyComponent.Width: = 35;
```

Приведенная строка устанавливает ширину (Width) компонента "MyComponent" в значение 35. Если Свойство Width компонента еще не было равно 35 к моменту выполнения данной строки программы, вы увидите, как компонента визуально изменит свою ширину.

Таким образом, нет ничего магического в Инспекторе Объектов. Object Inspector просто является удобным способом выполнения в

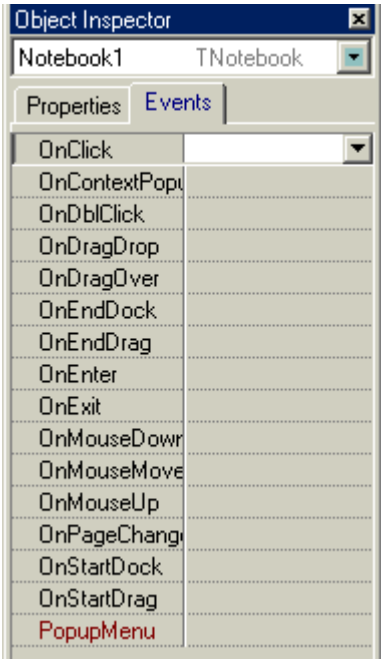

Рис.60. Вкладка События окна Инспектора объектов.

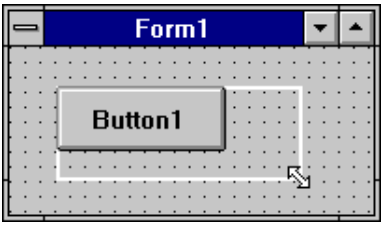

Рис.61. Кнопка на Форме.

режиме проектирования того, что может быть осуществлено программным путем в режиме выполнения (Run) приложения. Более того, как уже было сказано выше, у компонента могут быть Свойства, не отображаемые в окне Инспектора Объектов.

Объектно - ориентированный язык Паскаль, лежащий в основе Delphi, имеет принцип соответствия визуальных компонент тем вещам, которые они представляют. Разработчики Delphi поставили перед собой цель, чтобы, например, представление компонента Button (кнопка, рис.61), генерирующее некий код, соответствовало визуальному изображению кнопки на экране и являлось, как можно более близким эквивалентом реальной кнопки, которую вы можете найти на клавиатуре. И именно из этого принципа родилось понятие Свойства компонента.

Если вы измените Свойства Width и Height компонента Button, кнопка соответствующим образом изменит свои ширину и высоту. После изменения Свойства Width вам нет необходимости указывать объекту, чтобы он перерисовал себя, хотя при обычном программировании именно так вы и должны поступать.

Пример программы

Программа SHAPEDEM.DPR, окно которой изображено на рис.62, демонстрирует различные способы, с помощью которых можно изменять пользовательский интерфейс при выполнении программы. Эта

программа не производит никаких полезных действий, кроме демонстрации того, как легко можно создать "дельфийское" приложение (проект) с настраиваемым интерфейсом.

Программа SHAPEDEM содержит всего лишь объект TShape, размещенный на Форме, вместе с двумя полосами прокрутки и несколькими кнопками. Эта программа интересна тем, что позволяет в режиме выполнения (Run) изменять размер, цвет и внешний вид объекта TShape точно так же, как размер и цвет самой Формы. Листинг А показывает код программы SHAPEDEM (рис.62). Код головного модуля этой программы мы приведем по частям - по мере его написания.

Листинг А: Код головного модуля программы SHAPEDEM.DPR.

```
program Shapedem;
uses
Forms, Mina in 'MAIN.PAS' {Form1};
begin
Application.CreateForm(TForm1, Form1);
Application.Run;
end.
```

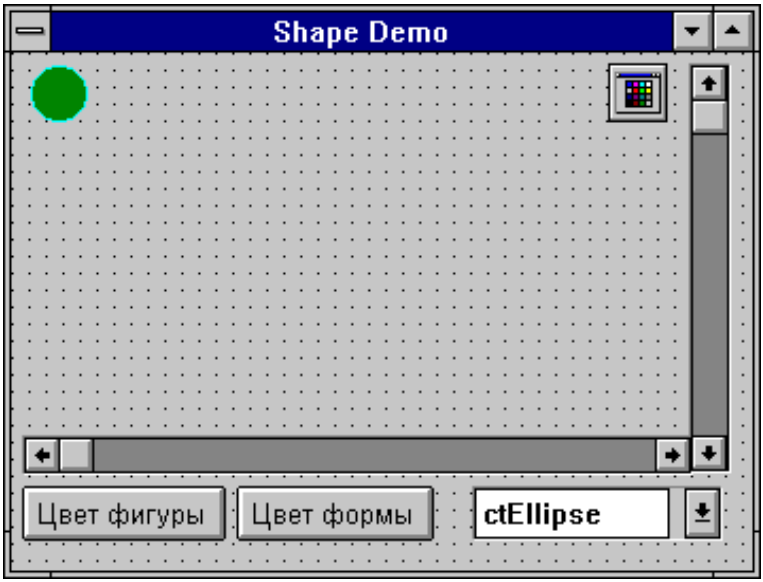

Рис.62. Программа SHAPEDEM.

В нашем примере полосы прокрутки (ScrollBars) используются для изменения размера фигуры, изображенной в средней части экрана. Для выбора нового вида фигуры используйте выпадающий список (ComboBox), а для изменения цвета фигуры или самого окна Формы используйте стандартное диалоговое окно выбора цвета, вызываемое кнопками "Цвет фигуры" и "Цвет Формы".

Для изменения "в режиме выполнения" цвета, какого - либо элемента или всего окна Формы достаточно выполнить всего лишь несколько действий. Для изменения цвета окна просто выберите компонент ColorDialog (справа вверху на рис.62) из палитры компонент, которая находится на страничке Dialogs, и поместите его на Форму.

Кроме того, поместите на Форму две обычные кнопки Button, которые находятся на страничке Standard. Для удобства чтения с помощью Object Inspector, измените имя компонента (Name) с Button1 (которое дается "по умолчанию") на FormColor, а его заголовок (Caption) на "Цвет Формы". Дважды щелкните по кнопке "Цвет Формы" и Delphi сгенерирует заготовку метода, который выглядит следующим образом:

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

Теперь введите сюда простые строчки кода:

```
procedure TForm1.FormColorClick(Sender: TObject);
begin
If ColorDialog1.Execute then Form1.Color: = ColorDialog1.Color;
end;
```

Во время выполнения программы, этот код, при нажатии кнопки "Цвет Формы" вызывает стандартное диалоговое окно выбора цвета, которое показано на рис.63. Если в этом диалоговом окне вы щелкните кнопку OK, выполнится следующая строка:

```
Form1.Color:=ColorDialog1.Color;
```

Этот код установит Свойство Color Формы Form1 в цвет, который был выбран с помощью диалогового окна ColorDialog1. Та же самая техника может использоваться для изменения цвета фигуры (компонент TShape). Все, что вам нужно сделать это поместить на Форму другую кнопку, изменить (при желании) ее Имя на ShapeColor, а Заголовок

- на "Цвет Фигуры", дважды щелкнуть по ней мышкой и создать метод аналогичный следующему:

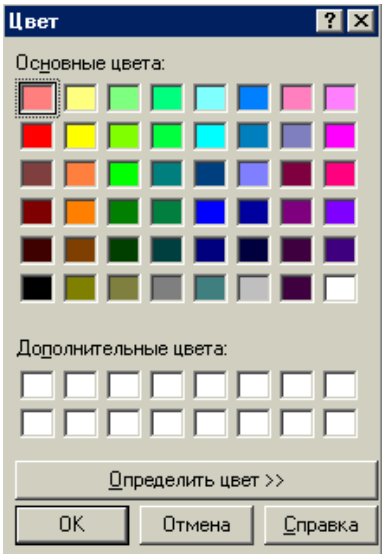

Рис.63. Выбор цвета.

```
procedure TForm1.ShapeColorClick (Sender: TObject);
begin
If ColorDialog1.Execute then Shape1.Brush.Color: = ColorDialog1.Color;
end;
```

Все эти действия можно проделать и автоматически, например, можно изменить цвет определенного элемента Формы, чтобы привлечь внимание пользователя, к какому - либо действию.

Весь механизм Windows - сообщений, используемый при взаимодействии компонент во время выполнения, оказывается скрытым от программиста, делая процесс создания программ наиболее легким. Сложное программирование в среде Windows становится доступным "широким" массам не профессиональных программистов. Например, программирование изменения размера фигуры с помощью полос прокрутки, требовавшее в "чистом" Windows сложной обработки сообщений в конструкции типа Case, в Delphi сводится к написанию одной единственной строчки кода.

Для начала, поместите два компонента ScrollBar на Форму (они находится на страничке Standard) и установите Свойство Kind первого компонента в sbHorizontal, а второго - в sbVertical. Переключитесь на страничку Events в Инспекторе Объектов и создайте заготовки метода для отклика на Событие OnChange для каждой полосы прокрутки. Напишите в каждом из методов по одной строчке кода следующим образом:

```
procedure TForm1.ScrollBar1Change(Sender: TObject);
begin
Shape1.Width: = ScrollBar1.Position * 3;
```

```
end;

procedure TForm1.ScrollBar2Change(Sender: TObject);
begin
Shape1.Height := ScrollBar2.Position * 2;
end;
```

Код программы, показанный здесь, устанавливает Свойства Width и Height фигуры TShape в соответствие с положением "бегунка" на полосах прокрутки (множители 3 и 2 введены только для лучшего представления).

Последняя часть кода программы SHAPEDEM демонстрирует большие возможности языка Object Pascal, на основе которого построена система Delphi. Вы можете ввести элементы в список компонента ComboBox, как в режиме проектирования, так и при выполнении программы. При этом в режиме проектирования вы можете просто ввести нужные элементы в список Items, щелкнув маленькую кнопку с тремя точками в правой части строки Инспектора Объектов, показывающей данное Свойство (Items).

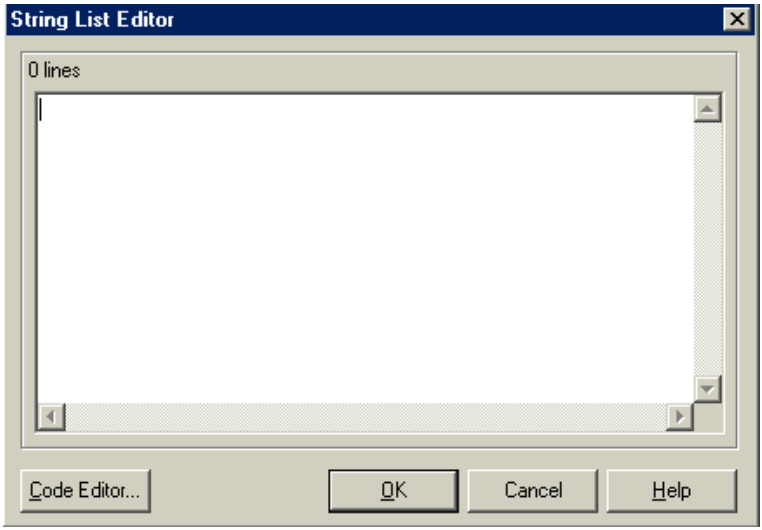

Рис.64. Редактор списка объекта ComboBox.

Перед вами появится диалоговое окно текстового редактора

(String List Editor), в котором вы и введете нужные элементы (рис.64). Другими словами, если вы выделите компонент Shape1 на Форме (просто щелкнув по нему) и посмотрите Свойство Shape в Инспекторе Объектов, то увидите список возможных видов фигур, которые может принимать данный компонент. Это, как раз те самые виды фигур, которые мы перечисляли в списке у компонента ComboBox1. Этот список вы можете найти в On - Line справочнике Delphi по контексту TShapeType. Если вы заглянете в исходный код класса TShape, то увидите те же элементы, формирующие перечислимый тип TShapeType:

```
TShapeType = (stRectangle, stSquare,
tRoundRect, stRoundSquare, stEllipse, stCircle);
```

Итак, смысл всего сказанного в том, что за всеми объектами, которые вы видите в "дельфийской" программе, стоит некий код на Паскале, к которому вы имеете доступ при "прямом" программировании. Это значит, что вы можете изменить поведение любой части вашей программы во время выполнения (Run) путем написания соответствующего кода.

В нашем конкретном случае, вам нужно написать только одну строчку кода, которая будет выполнена в качестве отклика на щелчок пользователем по выпадающему списку ComboBox1. Чтобы написать код этого отклика, в режиме проектирования выделите компонент ComboBox1 на Форме (как всегда, просто щелкнув по нему левой кнопкой мыши), а затем перейдите на страничку Events в Инспекторе Объектов. Дважды щелкните по пустому полю напротив События OnClick. В редакторе автоматически сгенерируется следующая заготовка метода:

```
procedure TForm1.ComboBox1Click(Sender: TObject);
begin
end;
```

Теперь вставьте одну строчку кода, чтобы метод выглядел следующим образом:

```
procedure TForm1.ComboBox1Click(Sender: TObject);
begin
Shape1.Shape: = TShapeType(ComboBox1.ItemIndex);
end;
```

Эта строчка кода устанавливает Свойство Shape компонента Shape1 в вид, который пользователь выберет в выпадающем списке. Этот код работает благодаря соответствию между порядковыми членами перечислимого типа и числовыми значениями различных элементов в ComboBox. Другими словами, первый элемент перечислимого типа имеет значение 0, что соответствует первому элементу, показанному в ComboBox (рис.64).

Давайте рассмотрим этот подход несколько подробней. Если вы рассмотрите декларацию перечислимого типа TShapeType, то увидите, что первый его элемент называется stRectangle. По определению, компилятор назначает этому элементу порядковый номер 0. Следующему по порядку элементу назначается номер 1 и т.д. Таким образом, слова stRectangle, stSquare и т.д., в действительности, просто символизируют порядковые номера в данном перечислимом типе.

На элементы в списке ComboBox также можно сослаться по их порядковому номеру, начиная с 0. Именно поэтому так важно (в данном случае) вводить указанные строки в строгом соответствии с декларацией типа TShapeType. Таким образом, используя преобразование типа TShapeType(ComboBox1.ItemIndex), вы можете указать компилятору, что общего имеют элементы в ComboBox и перечислимый тип в TShapeType, а именно, порядковые номера.

МЕТОДЫ DELPHI

Чтобы полностью понять и почувствовать все преимущества Delphi, вам нужно хорошо изучить язык Object Pascal. И хотя возможности визуальной части Delphi огромны, хорошим программистом может стать только тот, кто хорошо разбирается в технике ручного написания кода программы.

По мере обсуждения темы данной главы мы рассмотрим несколько простых примеров, которые, тем не менее, демонстрируют технику использования важных управляющих элементов системы Windows.

Создание методов

В предыдущей главе мы видели, что синтаксический "скелет" любого Метода может быть сгенерирован с помощью визуальных средств. Для этого нужно дважды щелкнуть мышкой в Инспекторе Объектов по пустой строчке напротив названия интересующего вас События для выделенного компонента. Заметим, если эта строчка не пуста, то двойной щелчок по ней просто переместит вас в окно Редактора Кода, в то место, где находится данный Метод.

Метод это процедура, которая определена, как часть класса и содержится в нем. Методы манипулируют полями и Свойствами классов (хотя могут работать и с любыми другими переменными). Доступ к полям и Методам других классов зависит от уровня "защищенности" этих полей и Методов. Пока же для нас важно то, что Методы можно создавать, как визуальными средствами, так и путем написания кода вручную.

Давайте рассмотрим процесс создания программы CONTROL1, которая поможет нам изучить технику написания методов в системе Delphi.

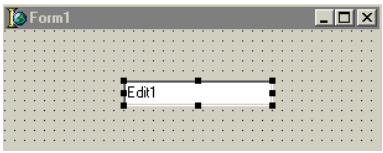

Рис.65. Компонент Edit.

Для создания программы CONTROL1 поместите с помощью мышки компонент Edit (он находится на страничке Standard Палитры Компонентов) на Форму. После этого ваша Форма будет иметь вид, показанный на рис.65.

Теперь перейдите в Object Inspector, выберите страничку Events (События) и дважды щелкните в пустой строчке напротив События

OnDblClick (Двойной щелчок), которое показано на рис.66. После этого, в активизировавшемся окне Редактора, вы увидите сгенерированный "скелет" метода Edit1DblClick, являющегося реакцией на Событие OnDblClick:

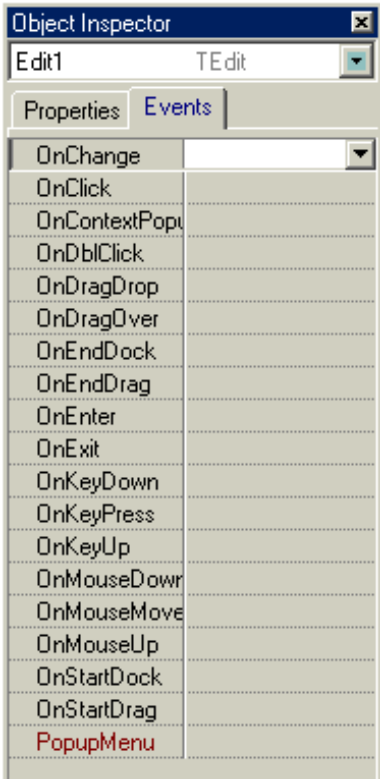

Рис.66. Свойства компонента Edit.

```
procedure TForm1.
Edit1DblClick (Sender: TObject);
begin
end;
```

После генерации заготовки процедуры вы можете оставить ее имя таким, каким "установил" его Delphi или изменить на любое другое (для этого просто введите новое имя в указанной выше строке Инспектора Объектов справа от требуемого События и нажмите Enter).

Теперь в окне Редактора Кода введите смысловую часть метода:

```
procedure TForm1.
Edit1DblClick (Sender: TObject);
begin
Edit1.Text: = 'Вы дважды щелкнули в строке редактирования';
end;
```

и сохраните программу. Во время выполнения программы дважды щелкните по строке редактирования. Текст в этой строке изменится в соответствии с тем, что мы написали в методе Edit1DblClick.

Листинг - А. Программа CONTROL1 демонстрирует, как создавать и использовать методы в Delphi.

```
program Control1;
uses
```

```
Forms, Main in 'MAIN.PAS' {Form1};
begin
Application.CreateForm(TForm1, Form1);
Application.Run;
end.
```

Листинг - В. Головной модуль (unit) программы CONTROL1.

```
unit Main;
interface
uses
WinTypes, WinProcs, Classes, Graphics, Controls, Printers,
Menus,
Forms, StdCtrls;
type
TForm1 = class(TForm)
Edit1: TEdit;
procedure Edit1DblClick(Sender: TObject);
end;
var
Form1: TForm1;
implementation
{$R *.DFM}

procedure TForm1.Edit1DblClick(Sender: TObject);
begin
Edit1.Text: = 'Вы дважды щелкнули в строке редактирования';
end;
end.
```

После того, как ваша программа загрузится в память, выполняются две строчки кода в CONTROL1.DPR, автоматически сгенерированные компилятором:

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

Первая строка запрашивает память у операционной системы и создает там объект Form1, являющийся экземпляром класса TForm1. Вторая строка указывает объекту Application, "по умолчанию" декларированному в Delphi, чтобы он запустил на выполнение главную

Форму приложения (Application). Сейчас мы не будем подробно останавливаться на классе TApplication и на автоматически создаваемом его экземпляре Application. Важно понять, что главное его предназначение - быть неким ядром, управляющим выполнением вашей программы.

Как правило, у большинства приведенных здесь примеров файлы проектов .DPR практически одинаковы. Поэтому в дальнейшем там, где они не отличаются кардинально друг от друга, мы не будем приводить их текст. Более того, в файл .DPR, автоматически генерируемый Delphi, в большинстве случаев нет необходимости заглядывать, поскольку все действия, производимые им, являются стандартными.

Итак, мы видели, что большинство кода Delphi генерирует автоматически. В большинстве приложений все, что вам остается сделать это вставить одну или несколько строк кода, как в методе Edit1DblClick:

```
Edit1.Text: = 'Вы дважды щелкнули в строке редактирования';
```

Каждая программа в Delphi состоит из файла проекта, имеющего расширение .DPR и одного или нескольких модулей, имеющих расширение .PAS. Модуль, в котором содержится главная Форма проекта, называется головным. Указанием компилятору о связях между разными модулями является предложение Uses, которое определяет зависимость модулей.

Нет никакого функционального различия между модулями, созданными вам в Редакторе и модулями, сгенерированными Delphi автоматически. В любом случае модуль подразделяется на три секции:

- Заголовок.
- Секция Interface - Интерфейс.
- Секция Implementation - Реализация.

и "скелет" любого модуля выглядит следующим образом (в фигурных скобках записываются комментарии):

```
unit Main; {Заголовок модуля}
interface {Секция Interface}
implementation {Секция Implementation}
end.
```

В интерфейсной секции (interface) модуля описывается все то, что

должно быть видимо для других модулей (типы, переменные, классы, константы, процедуры и функции). В секции implementation помещается код, реализующий классы, процедуры или функции, которые не видимы из других программ и модулей.

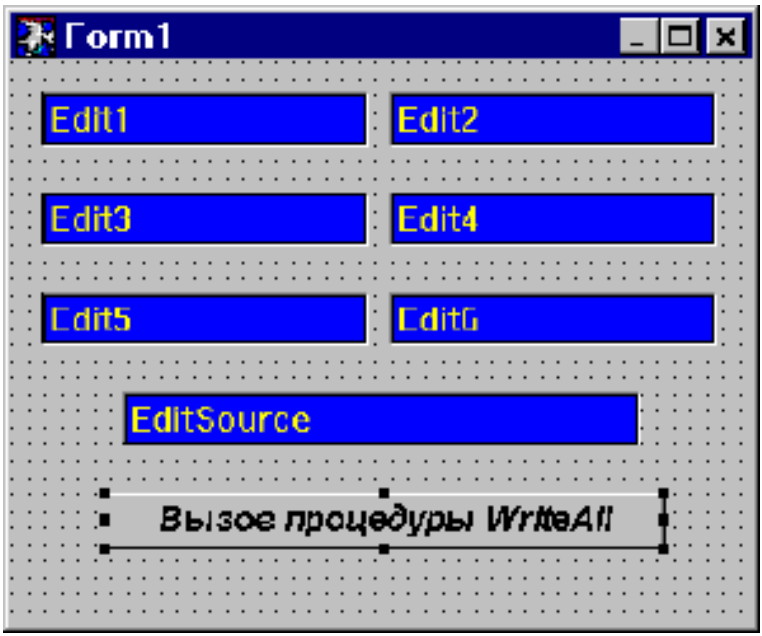

Рис.67. Вид окна программы Params.

Передача параметров

В Delphi процедурам и функциям (а, следовательно, и методам классов) могут передаваться параметры для того, чтобы обеспечить их необходимой для работы информацией. Программа PARAMS демонстрирует, как использовать передачу параметров в методы Delphi. Кроме того, мы узнаем, как создавать свои собственные процедуры, добавлять процедуру в класс, формируя метод класса, вызывать одну процедуру из другой.

Программа PARAMS позволяет вам вводить фразы в строки редактирования Edit. После нажатия кнопки "Вызов процедуры WriteAll" строка из управляющего элемента EditSource скопируется в шесть управляемых элементов - строк редактирования, как показано на

рис.67.

Далее мы не будем подробно останавливаться на том, как размещать компоненты на Форме - считаем, что это вы уже поняли и умеете делать. После того, как вы разместили на Форме семь компонентов Edit, переименуйте с помощью Инспектора Объектов седьмой компонент (Edit7) в EditSource. Поместите на Форму компонент Button и в Object Inspector измените его Заголовок (Caption) на "Вызов процедуры WriteAll" (естественно, вы можете заменить его шрифт, цвет и т.д.).

После завершения проектирования Формы класс TForm1 будет выглядеть следующим образом:

```
TForm1 = class(TForm)
Edit1: TEdit;
Edit2: TEdit;
Edit3: TEdit;
Edit4: TEdit;
Edit5: TEdit;
Edit6: TEdit;
EditSource: TEdit;
Button1: TButton;
end;
```

Следующий шаг состоит в добавлении метода, вызываемого по нажатию пользователем кнопки Button1. Это, напомним, можно сделать двумя способами:

- Перейти в Инспекторе Объектов на страничку Events (предварительно выделив компонент Button1 на Форме), выбрать слово OnClick и дважды щелкнуть мышкой на пустой строчке справа от него.
- Просто дважды щелкнуть на компоненте Button1 на Форме.

Delphi сгенерирует следующую "заготовку" кода:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
end;
```

Цель программы PARAMS - научить вас писать процедуры и передавать в них параметры. В частности, программа PARAMS реагирует на нажатие кнопки Button1 путем вызова процедуры WriteAll и передачи ей в качестве параметра содержимого строки редактирования

EditSource (EditSource.Text):

```
procedure TForm1.Button1Click(Sender: TObject);
begin
WriteAll(EditSource.Text);
end;
```

Важно понять, что объект EditSource является экземпляром класса TEdit и, следовательно, имеет Свойство Text, содержащее набранный в строке редактирования текст. Как вы уже, наверное, успели заметить, "по умолчанию" Свойство Text содержит значение, совпадающее со значением имени компонента (Name) - в данном случае это EditSource. Свойство Text вы, естественно, можете редактировать, как в режиме проектирования, так и во время выполнения программы.

Текст, который должен быть отображен в шести строках редактирования, передается процедуре WriteAll, как параметр. Чтобы передать параметр процедуре, просто напишите имя этой процедуры и заключите передаваемый параметр (параметры) в скобки:

```
WriteAll(EditSource.Text);
```

Заголовок этой процедуры выглядит следующим образом:

```
procedure TForm1.WriteAll(NewString: String);
```

где указано, что передаваемый процедуре параметр NewString должен иметь тип String - символьная строка. Вспомним, что задача процедуры WriteAll состоит в копировании содержимого строки редактирования EditSource в шесть других строк редактирования Edit1 - Edit6. Поэтому сама процедура должна выглядеть следующим образом:

```
procedure TForm1.WriteAll(NewString: String);
begin
Edit1.Text: = NewString;
Edit2.Text: = NewString;
Edit3.Text: = NewString;
Edit4.Text: = NewString;
Edit5.Text: = NewString;
Edit6.Text: = NewString;
end;
```

Поскольку процедура WriteAll не является откликом на какое -

либо Событие в Delphi, то ее нужно полностью написать "вручную". Простейший способ сделать это - скопировать Заголовок, какой - либо уже имеющейся процедуры, исправить его, а затем дописать необходимый код. Еще раз вернемся к заголовку процедуры - он состоит из пяти частей:

```
procedure TForm1.WriteAll(NewString: String);
```

Первая часть - зарезервированное слово "procedure", пятая часть - концевая точка с запятой ";". Обе эти части служат определенным синтаксическим целям, а именно:

- Первая информирует компилятор о том, что определен синтаксический блок "процедура".
- Вторая указывает на окончание заголовка (собственно говоря, все операторы в Delphi должны заканчиваться точкой с запятой).

Вторая часть заголовка - слово TForm1, квалифицирует то обстоятельство, что данная процедура является методом класса TForm1.

Третья часть заголовка - имя процедуры. Вы можете выбрать его любым, по вашему усмотрению. В данном случае мы назвали процедуру WriteAll.

Четвертая часть заголовка - параметр. Параметр декларируется внутри скобок и, в свою очередь, состоит из двух частей. Первая часть это собственно имя параметра, а вторая - его тип. Эти части разделены двоеточием. Если вы описываете в процедуре более чем один параметр, нужно разделить их точкой с запятой, например:

```
procedure Example(Param1: String; Param2: String);
```

После того, как вы "вручную" создали заголовок процедуры, являющейся методом класса, вы должны включить его в декларацию класса, например, путем копирования (еще раз напомним, что для методов, являющихся откликами на "дельфийские" События, данное включение производится автоматически):

```
TForm1 = class(TForm)
Edit1: TEdit;
Edit2: TEdit;
Edit3: TEdit;
Edit4: TEdit;
```

```
    Edit5: TEdit;
    Edit6: TEdit;
    EditSource: TEdit;
    Button1: TButton;

    procedure Button1Click(Sender: TObject);
    procedure WriteAll(NewString: String);
    end;
```

В данном месте нет необходимости оставлять в заголовке метода слово TForm1, так как оно уже присутствует в описании класса. Теперь приведем полный текст головного модуля программы PARAMS. Мы не включили сюда файл проекта, поскольку, как уже упоминалось, он практически одинаков для всех программ.

Листинг - С. Исходный код головного модуля программы PARAMS показывает, как использовать строки редактирования и как передавать параметры.

```
    Unit Main;
    interface
    uses WinTypes, WinProcs, Classes,Graphics, Controls, Printers,
Forms, StdCtrls;
    type
    TForm1 = class(TForm)
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    Edit4: TEdit;
    Edit5: TEdit;
    Edit6: TEdit;
    EditSource: TEdit;
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
    procedure WriteAll(NewString: String);
    end;
    var
    Form1: TForm1;
    implementation
    {$R *.DFM}
```

```
procedure TForm1.WriteAll(NewString: String);
begin
Edit1.Text := NewString;
Edit2.Text := NewString;
Edit3.Text := NewString;
Edit4.Text := NewString;
Edit5.Text := NewString;
Edit6.Text := NewString;
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
WriteAll(EditSource.Text);
end;
end.
```

При экспериментах с программой PARAMS вы можете попробовать изменить имена процедур и параметров. Однако следует помнить, что ряд слов в Delphi являются зарезервированными и употреблять их в идентификаторах (именах процедур, функций, переменных, типов и констант) не разрешается - компилятор сразу же обнаружит ошибку. К ним относятся такие слова, как procedure, string, begin, end и т.д. Полный их список приведен в On - Line справочнике Delphi.

Не старайтесь запомнить сразу все зарезервированные слова - компилятор "напомнит" вам о неправильном их использовании выдачей сообщения типа Identifier expected - Ожидался идентификатор, а обнаружено зарезервированное слово.

Методы и управляющие элементы

Теперь, когда вы освоили базовые понятия в системе программирования Delphi, можно продолжить изучение компонент и способов создания их методов. В программе CONTROL1, рассмотренной в предыдущем параграфе, был сгенерирован метод, являющийся откликом на Событие OnDblClick (двойной щелчок) строки редактирования Edit. Аналогично, можно сгенерировать метод, являющийся реакцией на Событие OnClick (щелчок).

В программе CONTROL2 расширен список находящихся на Форме компонентов и для многих из них определены События OnClick и OnDblClick. Для создания новой программы, можно просто скопировать файлы проекта CONTROL1 в новую директорию CONTROL2, изменить имя проекта на CONTROL2.DPR (в этом файле после ключе-

вого слова "program" также должно стоять название CONTROL2) и добавить компоненты Label, GroupBox, CheckBox, RadioButton и Button на Форму (эти компоненты находятся на страничке Standard Палитры Компонентов). Ваша Форма будет иметь вид, показанный на рис.68.

Заметим, что вы должны поместить компонент GroupBox (Панель группы) на Форму до того, как вы добавите компоненты CheckBox и RadioButton, которые, в нашем примере, должны быть "внутри" группового элемента. Иначе, объекты CheckBox1, CheckBox2, RadioButton1 и RadioButton2 будут "думать", что их родителем является Форма Form1 и при перемещении GroupBox1 по Форме, не будут перемещаться вместе с ней.

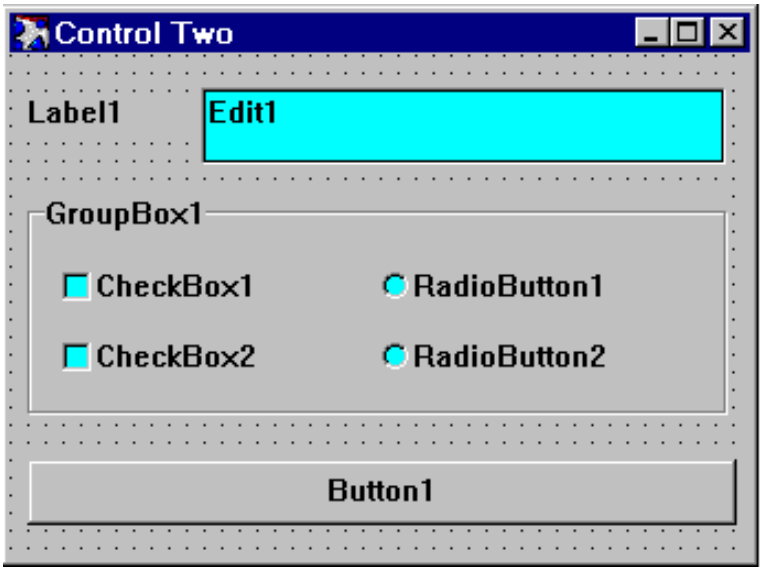

Рис.68. Окно программы Control2.

Таким образом, во избежание проблем, компонент, который должен быть "родителем" других компонент (Panel, GroupBox, NoteBook, StringGrid, ScrollBox и т.д.), нужно помещать на Форму до помещения на него его "детей" (CheckBox, RadioButton и т.д.).

Если вы все же забыли об этом и поместили "родителя" (например, GroupBox) на Форму после размещения на ней его "потомков" (например, CheckBox и RadioButton) - не отчаивайтесь. Отметьте (выделите) все необходимые объекты и скопируйте (с удалением) их в

буфер обмена Windows с помощью команд Главного меню Edit, Cut. После этого поместите на Форму и выделите нужный вам объект (например, GroupBox) и выполните команды Главного меню Edit, Paste. После этого все выделенные ранее объекты будут помещены на Форму и их "родителем" будет GroupBox. Описанный механизм является стандартным и может быть использован для всех видимых компонент.

Выберите объект Label1 и создайте для него Метод, являющийся откликом на Событие OnDblClick. Для этого введите в Метод одну строчку кода, например:

```
procedure TForm1.Label1DblClick(Sender: TObject);
begin
Edit1.Text: = 'Двойной щелчок на Label1';
end;
```

Запустите программу на выполнение, и дважды щелкните мышкой на метке Label1. Вы увидите, что строка редактирования изменится и в ней появится текст "Двойной щелчок на Label1".

Теперь закройте приложение и возвратитесь в режим проектирования. Добавьте обработчики Событий OnClick и OnDblClick для каждого объекта, имеющегося на Форме. Текст вашего головного модуля будет выглядеть следующим образом:

Листинг - D. Головной модуль программы CONTROL2.

```
Unit Main;
interface
uses WinTypes, WinProcs, Classes,Graphics, Controls,
StdCtrls, Printers, Menus, Forms;
type
TForm1 = class(TForm)
Label1: TLabel;
Edit1: TEdit;
Button1: TButton;
GroupBox1: TGroupBox;
CheckBox1: TCheckBox;
CheckBox2: TCheckBox;
RadioButton1: TRadioButton;
RadioButton2: TRadioButton;
procedure Edit1DblClick(Sender: TObject);
procedure Label1DblClick(Sender: TObject);
```

```
procedure CheckBox1Click(Sender: TObject);
procedure CheckBox2Click(Sender: TObject);
procedure RadioButton1Click(Sender: TObject);
procedure RadioButton2Click(Sender: TObject);
procedure Button1Click(Sender: TObject);
end;
var
Form1: TForm1;
implementation
{$R *.DFM}

procedure TForm1.Edit1DblClick(Sender: TObject);
begin
Edit1.Text:= 'Двойной щелчок на Edit1';
end;

procedure TForm1.Label1DblClick(Sender: TObject);
begin
Edit1.Text:= 'Двойной щелчок на Label1';
end;

procedure TForm1.CheckBox1Click(Sender: TObject);
begin
Edit1.Text:= 'Щелчок на CheckBox1';
end;

procedure TForm1.CheckBox2Click(Sender: TObject);
begin
Edit1.Text:= 'Щелчок на CheckBox2';
end;

procedure TForm1.RadioButton1Click(Sender: TObject);
begin
Edit1.Text:= 'Щелчок на RadioButton1';
end;

procedure TForm1.RadioButton2Click(Sender: TObject);
begin
Edit1.Text:= 'Щелчок на Radiobutton2';
end;
```

```
procedure TForm1.Button1Click(Sender: TObject);
begin
Edit1.Text:= 'Щелчок на Button1';
end;
end.
```

Эта программа служит двум целям:

- Она показывает, как создавать процедуры (методы) и как "наполнять" их содержательной "начинкой".
- Она демонстрирует технику работы с управляющими элементами Windows.

Как вы, наверное, заметили, методы программы CONTROL2, являющиеся откликами на События OnClick и OnDblClick, во многом похожи друг на друга.

СОБЫТИЯ DELPHI

Одна из ключевых целей среды визуального программирования (такой, как Delphi) - как бы скрыть от пользователя сложность программирования в системе Windows. При этом требуется, чтобы такая среда не была упрощена настолько, что программисты потеряют доступ к самой операционной системе.

Программирование, ориентированное на События - неотъемлемая и характерная черта Windows. Некоторые программные среды для быстрой разработки приложений (RAD), пытаются вообще скрыть от пользователя эту черту, как будто она настолько сложна, что большинство из них не смогут ее понять. Истина, по - видимому, заключается в том, что Событийное программирование, само по себе, не такое уж сложное. Однако есть некоторые особенности воплощения данной концепции, которые в некоторых ситуациях могут вызвать затруднения.

Среда Delphi дает полный доступ к подструктуре Событий, которые предоставляются системой Windows. С другой стороны, Delphi существенно упрощает программирование обработчиков таких Событий. В данной главе мы приведем несколько примеров того, как обрабатывать События в Delphi и дадим более детальное объяснение работы системы, ориентированной на События.

События

Объекты из библиотеки визуальных компонент (VCL) Delphi, так же, как и объекты нашего реального мира, имеют свой набор Свойств и свое поведение - набор откликов на События, происходящие с ними. Список Событий для данного объекта, на которые он реагирует, можно посмотреть, например, в Инспекторе Объектов на странице Событий.

На этой странице представлен список Событий, которые имеют тип, вроде TMouseMoveEvent (Событие движения мышки) и представляют собой процедуры - обработчики Событий. Например, OnDblClick соответствует двойному щелчку мыши, а OnKeyUp - Событию, когда нажатая клавиша была отпущена. Среди набора Событий для различных объектов из VCL есть, как События, импортируемые из Windows (MouseMove, KeyDown), так и События, порождаемые непосредственно в программе (например, DataChange для объекта TDataSource).

Поведение объекта определяется тем, какие обработчики и для каких Событий он имеет. Создание приложения в Delphi состоит из настройки Свойств используемых объектов и создания обработчиков

Рис.69. Диалог, появляющийся при щелчке мыши на Форме.

Событий.

Простейшие События, на которые иногда нужно реагировать - это, например, События, связанные с мышкой (они есть практически у всех видимых объектов) или Событие Click (Нажатие или щелчок по кнопке) для кнопки TButton.

Предположим, что вы хотите перехватить щелчок (среагировать на щелчок) левой кнопкой мышки на Форме. Чтобы сделать это - создайте новый проект, в Инспекторе Объектов выберите страницу Событий (Events) и сделайте двойной щелчок (левой кнопкой мышки) на правой части поля События OnClick. Вы получите следующую заготовку для обработчика данного События:

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

Напишите здесь следующий код:

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

Теперь, каждый раз, когда делается щелчок левой кнопки мыши по Форме, будет появляться окно диалога, показанное на рис.69.

Код, приведенный выше, представляет собой простейший случай ответа на Событие в программе на Delphi. Он настолько прост, что многие программисты могут написать такой код и без понимания того, что они на самом деле отвечают на сообщение о Событии, посланное им операционной системой. Хотя программист получает это Событие через "третьи руки", тем не менее, он на него отвечает.

Опытные программисты в Windows знают, что при возникновении События, операционная система передает вам не только уведомление о нем, но и некоторую связанную с ним информацию. Например, при возникновении События "нажата левая кнопка мыши" (OnMouseDown)

программа может информировать вас о том, в каком месте это произошло. Если вы хотите получить доступ к такой информации, то должны вернуться в Инспектор Объектов и создать обработчик События OnMouseDown:

```
procedure TForm1. FormMouseDown (Sender: TObject; Button:
TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
Canvas.TextOut(X, Y, 'X='+IntToStr(X)+'Y='+IntToStr(Y));
end;
```

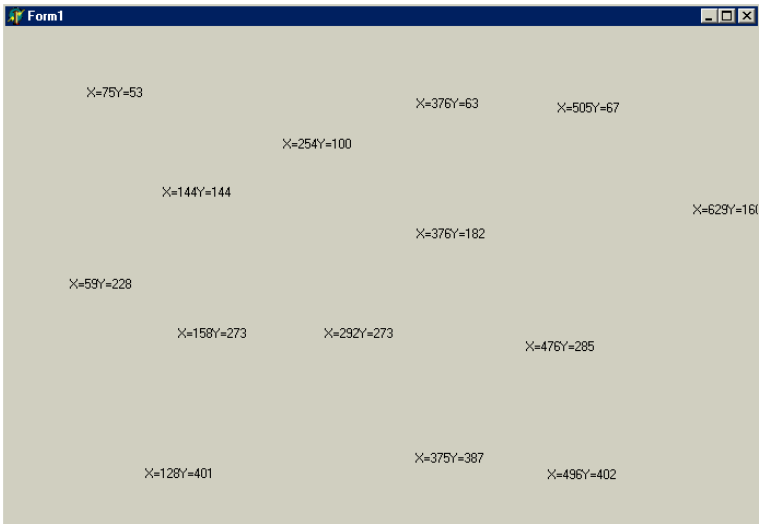

Рис.70. Окно программы с выводом координат щелчка мышкой.

Запустите программу и пощелкайте мышкой в разных местах Формы (рис.70) - на Форме появятся координаты ваших щелчков.

Как видите, в Delphi очень просто отвечать на События и не только на События, связанные с мышкой. Например, можно создать обработчик для OnKeyDown (Нажата клавиша):

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key:
Word;Shift: TShiftState);
begin
```

```
MessageDlg(Chr(Key), mtInformation, [mbOk], 0);
end;
```

Тогда на экран будет выводиться сообщение - какая клавиша нажата (рис.71).

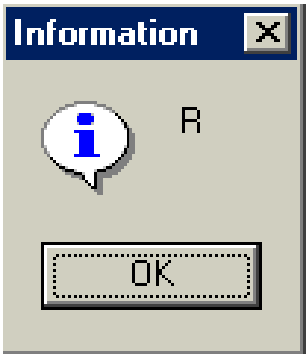

Рис.71. Сообщение о нажатой клавише.

Теперь, когда вы имеете некоторые знания о программировании Событий можно посмотреть, как полностью использовать имеющиеся возможности.

Обработка сообщений

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

Общий синтаксис для декларации обработчика сообщений Windows:

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

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

Например, при нажатии правой кнопки мыши на Форме в программе появляется всплывающее меню (Pop - Up Menu, если оно было привязано к этой Форме). Программист может захотеть привязать к правой кнопке какое - нибудь другое Событие. Это можно сделать так:

```
type
TForm1 = class(TForm)
PopupMenu1: TPopupMenu;
MenuItem1: TMenuItem;
MenuItem2: TMenuItem;
MenuItem3: TMenuItem;
private { Private declarations }

procedure WMRButtonDown(var Msg: TWMMouse);
Message WM_RBUTTONDOWN;

public { Public declarations }
end;
```

Далее, в секции implementation нужно написать обработчик События:

```
procedure TForm1.WMRButtonDown(var Msg: TWMMouse);
begin
MessageDlg('Right mouse button click.', mtInformation, [mbOK],
0);
end;
```

В данном случае при нажатии правой кнопки мыши будет появляться диалог - "Right mouse button click". У класса TForm уже есть унаследованный от дальнего предка обработчик данного События, который называется точно также и вызывает то самое Pop - Up меню. Если в новом обработчике сообщения нужно выполнить действия, которые производились в старом, то для этого применяется ключевое слово inherited. Если слегка модифицировать наш обработчик, то будет появляться Pop - Up меню:

```
procedure TForm1.WMRButtonDown (var Msg: TWMMouse);
begin
MessageDlg('Right mouse button click.', mtInformation, [mbOK],
0);
Inherited;
end;
```

Однако есть еще способ обработки всех сообщений, которые получает приложение. Для этого используется Свойство OnMessage объ-

екта Application, который автоматически создается при запуске программы. Если определен обработчик События OnMessage, то он получает управление при любом Событии, сообщение о котором направлено в программу. Следующий код, будет приводить к появлению на экране диалога при двойном щелчке мышки на любом объекте в приложении:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
Application.OnMessage:=AOM;
end;

procedure TForm1.AOM(var Msg: TMsg; var Handled: Boolean);
begin
Handled:=False;
If Msg.Message = WM_LBUTTONDBLCLK then begin
MessageDlg('Double click.', mtInformation, [mbOK], 0);
Handled:=True;
end;
```

Конечно, в обработчике событий нельзя выполнять операции, требующие длительного времени, поскольку это приведет к замедлению выполнения всего приложения - вашего проекта Delphi.

ЗАКЛЮЧЕНИЕ

На этом мы заканчиваем краткий обзор основных методов и приемов программирования на алгоритмическом языке Turbo Pascal - 6.0 и работы в системе разработки пользовательских приложений Delphi - 6.0. Вы получили начальное представление о самой системе Delphi, некоторых методах работы с ней и основных настройках всей среды.

Были рассмотрены все основные понятия, используемые в Delphi - Свойства, Методы и События, приведено несколько примеров программ, которые реально сделаны на Delphi и могут быть повторены вами.

Мы надеемся, что данная книга поможет вам освоить основные приемы программирования на языке Pascal и работы в системе Delphi.

ПРИЛОЖЕНИЕ

Инсталляция среды Turbo Pascal

Система Turbo Pascal поставляется с программой инсталляции всей среды, которая находится в файле INSTALL.EXE. Эту программу можно использовать для установки Turbo Pascal на вашем компьютере. Программа автоматически установит вам все файлы системы в нужные места на жестком диске.

Вначале вам необходимо использовать команду DISKCOPY (или другой способ) для создания рабочих копий дистрибутивных дисков - дискет. Чтобы инсталлировать Turbo Pascal на жестком диске вашего компьютера выполните следующие действия:

- Вставьте рабочую дискету с дистрибутивами в дисковод A: .
- Наберите в командной строке (если вы работаете в системе DOS) команду A:\INSTALL и нажмите Enter. Если вы работаете с оболочкой Norton Commander перейдите на диск A: и запустите файл INSTAL.EXE.
- Нажмите клавишу Enter для вывода экрана инсталляции системы.
- Следуйте далее подсказкам, которые появляются на экране вашего компьютера.

Когда инсталляция закончится, программа INSTALL предложит вам прочитать файл ReadMe, который содержит последнюю информацию о данной версии системы. INSTALL так же сообщит вам, как конфигурировать файлы CONFIG.SYS и AUTOEXEC.BAT для использования Turbo Pascal.

Сразу после инсталляции можно использовать справочно - обучающую систему TPTOUR, которая приводит полное описание методов работы с интегрированной средой Turbo Pascal. По умолчанию TPTOUR инсталлируется в справочнике - папке TOUR. Далее вы можете начать работать с самой интегрированной средой разработки (IDE) Turbo Pascal. Для этого просто перейдите в справочник, в котором инсталлирован Turbo Pascal и наберите TURBO (если вы работаете в системе DOS) или запустите файл TURBO.EXE, если вы работаете с оболочкой Norton Commander.

Новая интегрированная среда позволяет вам выполнять все настройки системы без выхода из интерфейса программы. Можно также использовать опции командной строки, когда запускаете интегриро-

ванную среду в системе DOS. Если необходимо, чтобы IDE сохраняла и восстанавливала свою конфигурацию между сеансами работы, войдите в диалоговое окно Preferences (Главное меню Options, Environment) и включите опции Environment и Desktop.

ЛИТЕРАТУРА

1. Руководство пользователя Turbo Pascal. // Borland Co., 1980.
2. Фаронов В.В. - Основы Турбо Паскаля. // М., МВТУ, 1992.
3. Пильщиков В.Н. - Сборник упражнений по языку Паскаль. // М., Наука, 1989.
4. Гольденберг В.А. - Введение в программирование. // Минск, Харвест, 1997.
5. Попов В.Б. - Turbo Pascal для школьников. // М., ФиС, 1996.
6. Зубов В.С. - Программирование на языке Turbo Pascal. // М., Филин, 1997.
7. Андреева Е.В., Фалина И.Н. - Турбо - Паскаль в школе. // М., МГУ, 1998.
8. Джордейн Р. - Справочник программиста персональных компьютеров типа IBM PC, XT и AT, Пер. с англ. и предисл. Н.И. Гайского, М., Финансы и статистика, 1991.
9. Пярнпу А.А. - Программирование на современных алгоритмических языках. // М., Наука, 1990.
10. Очков В.Ф., Пухначев Ю.В. - 128 советов начинающему программисту. // М., Энергоатомиздат, 1991.
11. Попов Б.А., Теслер Г.С. - Вычисление функций на ЭВМ. // Киев, Наукова думка, 1984.
12. Смирнов Н.Н. - Программные средства персональных ЭВМ. // Л., 1990.
13. Шрайберг Я.Л. - Справочное руководство по основам Информатики. // М., Финансы и статистика, 1990.
14. Help файл программы Delphi - 6.0. // Borland Co., 2000.
15. Использована информация по Delphi с десятка Web - серверов в Интернете, авторы которой не указаны.
16. Использована информация по Delphi из файлов на CD -дисках, авторы которой не указаны.

Дубовиченко
Сергей Борисович

Член - корреспондент Казахстанской Международной
Академии Информатизации,
член Нью - Йоркской Академии Наук,
член Европейского Физического Общества

Программирование

Pascal и Delphi

Учебник по информатике
для ВУЗов

Подписано к печати 01.08.2003. Формат 60x84 1/16. Бумага офсетная.
Печать офсетная. Усл. изд. листов 17,7. Тираж 500. Цена договорная.

Издательство Казахско - Американского Университета, Казахстан,
Алматы, ул. Сатпаева 18а, тел. 64-46-10
Отпечатано в типографии Каз.ГАСА, Казахстан, Алматы,
тел. 20-09-45