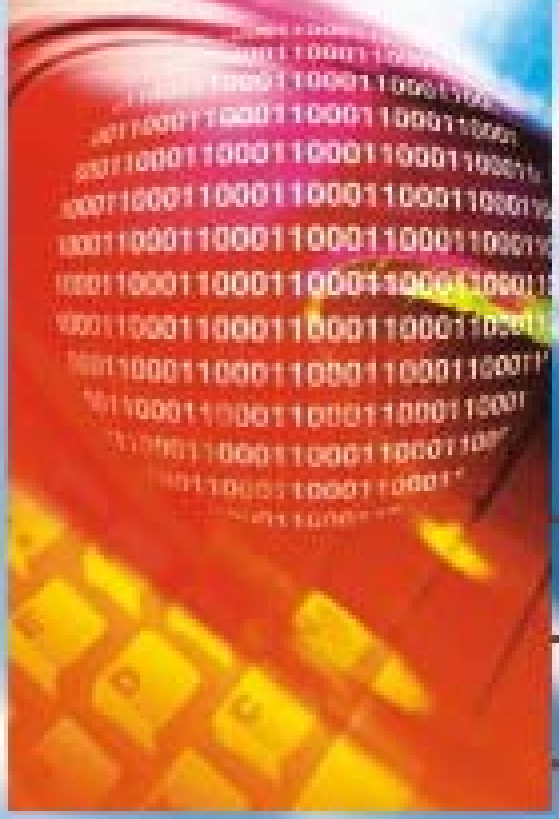
Евгений Марков
Петр Дарахвелидзе

Delphi 2005 для Win32



- Пользовательский интерфейс и бизнес-логика программирования
- Использование XML
- Современные технологии доступа к данным
- Распределенные многозвенные приложения
- Web-приложения и Web-службы

Наиболее полное руководство

+CD-ROM

В ПОДЛИННИКЕ®

УДК 681.3.068+800.92Delphi2005
ББК 32.973.26-018.1
Д20

Дарахвелидзе П. Г., Марков Е. П.

Д20 Delphi 2005 для Win32. — СПб.: БХВ-Петербург, 2005. — 1136 с.: ил.

ISBN 5-94157-700-1

Рассмотрены практические аспекты программирования в Borland Delphi 2005 для Win32. Приведено детальное описание объектной модели Delphi, обсуждаются вопросы разработки бизнес-логики приложений. Большое внимание уделено созданию приложения для работы с базами данных. Рассмотрены технологии доступа к данным Borland DataBase Engine, dbExpress, Interbase Express, dbGo, а также создание распределенных приложений баз данных и технология DataSnap. Показана работа с редактором отчетов Rave Report 6.0 и преобразование данных в формат XML. Описана разработка Internet-приложений, распределенные Web-приложения и Web-службы. Все рассматриваемые темы сопровождаются подробными примерами, исходные коды которых находятся на прилагаемом компакт-диске.

Для подготовленных программистов

УДК 681.3.068+800.92Delphi2005
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Леонид Кочин*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Наталия Першакова*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 31.08.05.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 91,59.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию № 77.99.02.953.Д.006421.11.04
от 11.11.2004 г. выдано Федеральной службой по надзору
в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ОАО "Техническая книга"
190005, Санкт-Петербург, Измайловский пр., 29.

ISBN 5-94157-700-1

Оглавление

Глава 2

Типы данных в Delphi

В этой главе рассматриваются основные типы данных, применяемые в Delphi, а также некоторые часто используемые конструкции языка. Знание основных типов данных облегчает программирование и позволяет избежать ряда ошибок начинающим программистам.

Все существующие в Delphi типы данных можно разделить на две основные группы. К первой относятся стандартные типы, предопределенные в среде разработки. На их основе разработчик может описывать собственные (определяемые) типы данных, которые относятся ко второй группе.

Определяемые типы данных существенно расширяют возможности программистов. Достаточно открыть любой файл с исходным кодом VCL, чтобы убедиться, что этот инструмент широко применяется разработчиками фирмы Borland. Однако при определении собственных типов данных требуется соблюдать осторожность и тщательно продумывать, какой предопределенный тип данных положить в основу собственного. Для этого необходимо иметь максимум информации о стандартных типах данных.

В главе рассматриваются следующие вопросы.

- Типы данных, используемые в Delphi.
- Тип вариант.
- Применение записей.
- Работа с массивами.

Материал главы рассчитан на читателя, имеющего представление о самом языке Delphi, его операторах и основных возможностях.

Простые типы

В эту категорию входят традиционные типы, большинство из которых сопровождают разработчиков чуть ли не с начала компьютерной эры. Сюда входят числовые, логический и символьный типы данных.

Несмотря на привычность и ординарность, в современной реализации простых типов данных имеется ряд особенностей, которые разработчик должен знать. В табл. 2.1 представлено описание простых типов, предназначенных для хранения чисел.

Таблица 2.1. *Простые числовые типы данных*

Тип данных	Значения	Число байтов в значении
`Integer`	–2147483648..2147483647	4 (со знаком)
`Cardinal`	0..4294967295	4
`ShortInt`	–128..127	1 (со знаком)
`SmallInt`	–32768..32767	2 (со знаком)
`LongInt`	–2147483648..2147483647	4 (со знаком)
`Int64`	$-2^{63}..2^{63}-1$	8 (со знаком)
`Byte`	0..255	1
`Word`	0..65535	2
`LongWord`	0..4294967295	4
`Real`	5.0×10^{-324} .. 1.7×10^{308}	8 (со знаком)
`Real48`	2.9×10^{-39} .. 1.7×10^{38}	6 (со знаком)
`Single`	1.5×10^{-45} .. 3.4×10^{38}	4 (со знаком)
`Double`	5.0×10^{-324} .. 1.7×10^{308}	8 (со знаком)
`Extended`	3.6×10^{-4951} .. 1.1×10^{4932}	10 (со знаком)
`Comp`	$-2^{63}+1$ .. $2^{63}-1$	8 (со знаком)
`Currency`	–922337203685477.5808.. 922337203685477.5807	8 (со знаком)

Для хранения целочисленных значений рекомендуется использовать базовые типы `Integer` (для положительных и отрицательных чисел) и `Cardinal` (для положительных чисел, вдвое больших `Integer`), так как они оптимизированы для выполнения вычислительных операций процессором.

Тип данных с плавающей точкой в языке Pascal появился, еще когда наличие сопроцессора (скажем, 80287) было скорее исключением, чем правилом. Все операции выполнялись в режиме эмуляции, для этого был выбран 6-байтовый (48-разрядный) формат, названный `Real`.

Сейчас процессоров без сопроцессора не бывает. Они работают со стандартизованными Американским институтом инженеров электротехники и электроники (IEEE) форматами: 80-разрядным (в Delphi — `Extended`), 64-разрядным (`Double`) и 32-разрядным (`Single`). Соответственно, старый добрый `Real` стал настоящим рудиментом, пожирающим массу времени на преобразование чисел.

На самом деле в Delphi под именем `Real` скрывается тип `Double`, что позволяет намного ускорить вычисления. Поэтому здесь необходимо учитывать следующие тонкости. Во-первых, размер всех массивов и структур с `Real` увеличится на 25%, что в среде Win32 не так уж страшно. Во-вторых, не работает код, в явном виде ориентированный на длину `Real` (6 байт). В этом случае можно описать переменные с новым типом `Real48`:

```
var r:real; r48:real48;
...
  ShowMessage(IntToStr(SizeOf(r)));
  ShowMessage(IntToStr(SizeOf(r48)));
...
```

В первом случае будет выдан результат 8, во втором — прежние 6.

Можно использовать директиву компилятора `{$REALCOMPATIBILITY ON}`, которая отменяет замену `Real` на `Double`. В этом случае описанный выше пример два раза выдаст ответ 6.

Для символьных данных (значение переменной данного типа хранит один символ) базовым является тип `Char`. На его основе созданы два дополнительных типа — `AnsiChar` и `WideChar`. Они обеспечивают работу с двумя кодировками символов: ANSI и Unicode.

Примечание

Кодировка ANSI отводит для хранения символа один байт. Кодировка Unicode является более сложной. В ней символы могут быть представлены одним или двумя байтами.

В Delphi расширены возможности логического типа данных. Помимо базового типа `Boolean` существуют дополнительные типы `ByteBool` (один байт), `WordBool` (два байта), `LongBool` (четыре байта), которые необходимы для обеспечения совместимости с функциями Windows и другими средами разработки (C++).

В базовом логическом типе все осталось по-прежнему: его возможные значения `True` (1) и `False` (0). В остальных типах значению `False` соответствует 0, всем остальным числам соответствует `True`.

Перечисляемый тип не претерпел никаких изменений. Он представляет собой упорядоченное множество значений:

```
type
    TSomeEnumeration = (seFirst, seSecond, seThrid);
var Enum1: TSomeEnumeration;
    Enum2: (Val1, Val2, Val3, Val4);
...
Enum1 := seSecond;
Enum2 := Val1;
...
```

Последний простой тип — поддиапазон — представляет собой подмножество любого другого базового простого типа:

```
type TOtherEnum = (Win95, NT4, Win98, Win2000, XP, Win 2003, LongHorn);
     TSub = 0..127;
var Sub1: TSub;
    Sub2: Win98..Win2000;
    Sub3: 'a'..'z';
...
Sub1 := 100;
Sub2 := NT4;
Sub3 := 'g';
...
```

Строковый тип

Самые разнообразные тексты встречаются в любых приложениях. Для их хранения можно применять константы и переменные. Для представления текстовых значений в Delphi есть три типа.

- Тип `ShortString` — может представлять текст длиной до 255 символов. Переменная этого типа имеет фиксированный размер 256 байтов. Первый байт отводится для хранения длины текста в символах. Остальные байты отведены для хранения символов.
- `AnsiString` — предназначен для создания переменных для хранения текстов произвольной длины. Тексты состоят из символов в кодировке ANSI (на один символ отводится один байт). Память для переменных этого типа выделяется динамически и ограничена только доступным адресным пространством. Сама переменная представляет собой указатель размером 4 байта. Если переменная пуста, указатель равен `Nil`. Две переменные,

имеющие одинаковые значения, ссылаются на одну область памяти (их указатели равны).

- Тип `WideString` — также осуществляет динамическое выделение памяти для хранения текста. Текст должен содержать символы в кодировке Unicode (два байта на символ).

Тип `ShortString` оставлен для обеспечения обратной совместимости с более ранними версиями. По умолчанию в проекте устанавливается директива компилятора `{$H+}`, которая определяет, что объявление

```
var SomeStr: String;
```

соответствует типу `AnsiString`. Директива `{$H-}` определяет, что такое объявление соответствует типу `ShortString`. Впрочем, можно употреблять типы `ShortString` и `AnsiString` напрямую.

Длину текста для строковой переменной можно ограничить числом символов, меньшим 255:

```
var SomeString[64];
```

К отдельному символу текста можно обратиться по его порядковому номеру (индексу):

```
var S: Char;
...
S := SomeString[4];
```

Помните, что для типа `ShortString` нулевой индекс соответствует одному байту длины текста.

Структурные типы

Структурные типы отличаются от простых тем, что переменная такого типа может содержать несколько значений простого типа одновременно. В зависимости от конкретного структурного типа это могут быть значения одного простого типа (множество, массив) или нескольких (запись).

Все структурные типы, кроме набора, могут объединять значения не только простых, но и структурных типов.

Данные в переменной структурного типа по умолчанию выравниваются по границе слова или двойного слова. Это сделано для ускорения доступа к данным.

Если переменная занимает много места, то быстротой можно пожертвовать ради экономии места — значения можно хранить упакованными. Для этого предназначено ключевое слово `packed`:

```
type TSomeStruct = packed set of 1..100;
```

Множества

Множество представляет собой набор значений одного из простых типов (кроме вещественного). Переменную-множество можно объявить через тип или напрямую:

```
type TSomeSet = set of Byte;
     TNumSet = 0..100;
var SetVar1: TSomeSet;
    SetVar2: set of 'A'..'Z';
```

В программе множеству можно присваивать произвольное подмножество:

```
SetVar1 := [1, 3, 5, 7, 11];
SetVar2 := ['X', 'Y', 'Z'];
```

Для контроля значений множеств имеется специальный оператор `in`:

```
if SetVar1 in [1, 3, 5]
  then ShowMessage('Простое число');
```

Массивы

В распоряжении программистов в Delphi есть массивы нескольких видов. Массивы могут быть открытыми, их размер теперь можно задавать динамически, также доступны многомерные динамические массивы. Однако обо всем по порядку.

Статические массивы

В Delphi нельзя при описании массива опустить ни нижнюю, ни верхнюю границы. Как правило, нумерация массивов начинается с нуля. Но если уж очень хочется начать нумерацию с единицы, теперь нужно написать:

```
var A: array[1..5] of integer;
```

При передаче массивов в качестве параметров процедур нельзя явно указывать границы. Следующее объявление функции

```
function atest(x: array[1..5] of integer): integer;
```

будет отвергнуто компилятором. Допускается только "безграничный" вариант:

```
function atest(x: array of integer): integer;
```

В Delphi существует строгая типизация. Так, в случае

```
Var
  A1: array[1..5] of integer;
  A2: array[1..5] of integer;
```

переменные A1 и A2 будут иметь разный тип! Естественно, массив A1 и формальный параметр функции Atest тоже будут иметь разный тип. Чтобы не нарушать канонов языка, массивы в функции передаются как динамические, т. е. вместе с адресом массива передается и число элементов в нем. Внутри тела функции можно получить значения граничных индексов с помощью функций Low() и High():

```
function atest(x: array of integer): integer;
var i: Integer;
begin
  Result := 0;
  for i:=Low(x) to High(x) do
    Result := Result + x[i];
end;
```

Есть и другой способ не отступить от канонов и остаться в рамках статических массивов — типизировать их:

```
type
 TMyArray = array[1..5] of integer;
Var
  A1: TMyArray;
function atest2(x: TMyArray): integer;
```

В этом случае компилятор не передает вызываемой функции длину массива — она ведь известна заранее. Это очень важно, если вы вызываете из программ на Delphi функции, написанные на других языках, — C, Fortran и т. д. Там "лишнее" двойное слово, содержащее длину массива, нарушит правильность работы. Поэтому присваивайте всем передаваемым массивам определенный тип.

Примечание

Обычно массивы передаются по ссылке, поэтому перед описанием таких параметров в функции следует указать ключевое слово var. Передача массивов, особенно больших, по значению (т. е. без ключевого слова var) может создать проблемы с памятью. Если вы боитесь употреблять тип var, а значения элементов не должны изменяться, опишите этот параметр с ключевым словом const.

Кстати, нумерация элементов массива не с 0 настолько несовременна, что может привести к курьезной ситуации. Передадим массив типа TMyArray в функцию, рассчитанную на динамический массив, и покажем значения границ:

```
function atest(x: array of integer): integer;
begin
```