Федеральное агентство по образованию

**Реферат**

**«ТИПЫ ДАННЫХ В ПАСКАЛЕ»**

2008

**1. Типы данных**

Любые данные, т.е. константы, переменные, свойства, значения функций или выражения характеризуются своими типами. Тип определяет множество допустимых значений, которые может иметь тот или иной объект, а также множество допустимых операций, которые применимы к нему. Кроме того, тип определяет также и формат внутреннего представления данных в памяти ПК.

Вообще язык Object Pascal характеризуется разветвленной структурой типов данных (рис. 1.1). В языке предусмотрен механизм создания новых типов, благодаря чему общее количество используемых в программе типов может быть сколь угодно большим.

Обрабатываемые в программе данные подразделяются на переменные, константы и литералы:

*Константы* представляют собой данные, значения которых установлены в разделе объявления констант и не изменяются в процессе выполнения программы.

*Переменные* объявляются в разделе объявления переменных, но в отличие от констант получают свои значения уже в процессе выполнения программы, причем допускается изменение этих значений. К константам и переменным можно обращаться по именам.

*Литерал* не имеет идентификатора и представляется в тексте программы непосредственно значением.

*Тип* определяет множество значений, которые могут принимать элементы данных, и совокупность допустимых над ними операций.

В этой и четырех последующих главах приводится подробное описание всех типов.

**1.1 Простые типы**

К простым типам относятся порядковые, вещественные типы и тип дата-время.

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

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

*Тип дата-время* предназначен для хранения даты и времени. Фактически для этих целей он использует вещественный формат.

**1.1.1 Порядковые типы**

К порядковым типам относятся (см. рис. 1.1) целые, логические, символьный, перечисляемый и тип-диапазон. К любому из них применима функция Ord(x), которая возвращает порядковый номер значения выражения X.

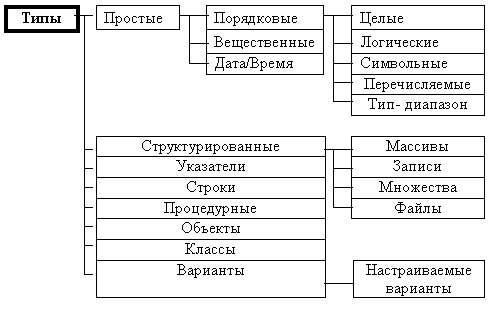


Рис. 1.1 - Структура типов данных

Для *целых* типов функция ord(x) возвращает само значение х, т. е. Ord(X) = х для х, принадлежащего любому *целому* типу. Применение Ord(x) к *логическому*, *символьному и перечисляемому* типам дает положительное целое число в диапазоне от 0 до 1 (*логический тип*), от 0 до 255 (*символьный*), от 0 до 65535 (*перечисляемый*). *Тип-диапазон* сохраняет все свойства базового порядкового типа, поэтому результат применения к нему функции ord(х) зависит от свойств этого типа.

К порядковым типам можно также применять функции:

*pred(x)* - возвращает предыдущее значение порядкового типа (значение, которое соответствует порядковому номеру ord (х) -1, т. е. оrd(рred(х)) = оrd(х) - 1;

*succ (х)* - возвращает следующее значение порядкового типа, которое соответствует порядковому номеру ord (х) +1, т. е. оrd(Succ(х)) = оrd(х) + 1.

Например, если в программе определена переменная

var

с : Char;

begin

с := '5';

end;

то функция PRED(с) вернет символ '4', а функция SUCC(с) - символ '6'.

Если представить себе любой порядковый тип как упорядоченное множество значений, возрастающих слева направо и занимающих на числовой оси некоторый отрезок, то функция pred(x) не определена для левого, a succ (х) - для правого конца этого отрезка.

*Целые типы*. Диапазон возможных значений целых типов зависит от их внутреннего представления, которое может занимать один, два, четыре или восемь байтов. В табл. 1.1 приводятся названия целых типов, длина их внутреннего представления в байтах и диапазон возможных значений.

Таблица 1.1 - Целые типы

|  |  |  |
| --- | --- | --- |
| Название | Длина, байт | Диапазон значений |
| Cardinal | 4 | 0. .. 2 147 483 647 |
| Byte | 1 | 0...255 |
| Shortint | 1 | -128...+127 |
| Smallint | 2 | -32 768...+32 767 |
| Word | 2 | 0...65 535 |
| Integer | 4 | -2 147 483 648...+2 147 483 647 |
| Longint | 4 | -2 147 483 648...+2 147 483 647 |
| Int64 | 8 | -9\*1018...+9\*1018 |
| LongWord | 4 | 0. . .4 294 967 295 |

Типы **LongWord** и **Int64** впервые введены в версии 4, а типы **Smallint** и **Cardinal** отсутствуют в Delphi 1. Тип **integer** для этой версии занимает 2 байта и имеет диапазон значений от -32768 до +32767, т. е. совпадает с **Smallint**.

При использовании процедур и функций с целочисленными параметрами следует руководствоваться “вложенностью” типов, т.е. везде, где может использоваться **word**, допускается использование **Byte** (но не наоборот), в **Longint** “входит” **Smallint**, который, в свою очередь, включает в себя **Shortint**.

Перечень процедур и функций, применимых к целочисленным типам, приведен в табл. 1.2. Буквами b, s, w, i, l обозначены выражения соответственно типа **Byte**, **Shortint, Word, Integer и Longint**,

х - выражение любого из этих типов; буквы vb, vs, vw, vi, vl, vx обозначают переменные соответствующих типов. В квадратных скобках указывается необязательный параметр.

Таблица 1.2 - Стандартные процедуры и функции, применимые к целым типам

|  |  |  |
| --- | --- | --- |
| Обращение | Тип результата | Действие |
| abs (x) | x | Возвращает модуль x |
| chr(b) | Char | Возвращает символ по его коду |
| dec (vx [, i] ) | - | Уменьшает значение vx на i, а при отсутствии i - на 1 |
| inc(vx[,i]) | - | Увеличивает значение vx на i, а при отсутствии i -на 1 |
| Hi(w) | Byte | Возвращает старший бант аргумента |
| Hi(I) | То же | Возвращает третий по счету байт |
| Lo(i) | “ | Возвращает младший байт аргумента |
| Lo(w) | “ | То же |
| odd(l) | Boolean | Возвращает True, если аргумент-нечетное число |
| Random(w) | Как у параметра | Возвращает псевдослучайное число, равномерно распределенное в диапазоне 0...(w-l) |
| sqr(x) | X | Возвращает квадрат аргумента |
| swap(i) | Integer | Меняет местами байты в слове |
| swap (w) | Word | Тоже |

При действиях с целыми числами тип результата будет соответствовать типу операндов, а если операнды относятся к различным целым типам - общему типу, который включает в себя оба операнда. Например, при действиях с **shortint** и **word** общим будет тип **integer**. В стандартной настройке компилятор Delphi не вырабатывает код, осуществляющий контроль за возможной проверкой выхода значения из допустимого диапазона, что может привести к недоразумениям.

*Логические типы*. К логическим относятся типы **Boolean, ByteBool, Bool, wordBool** и **LongBool**. В стандартном Паскале определен только тип **Boolean**, остальные логические типы введены в Object Pascal для совместимости с Windows: типы **Boolean** и **ByteBool** занимают по одному байту каждый, **Bool** и **WordBool** - по 2 байта, **LongBool** - 4 байта. Значениями логического типа может быть одна из предварительно объявленных констант False (ложь) или True (истина).

Поскольку логический тип относится к порядковым типам, его можно использовать в операторе цикла счетного типа. В Delphi 32 для **Boolean** значение

Ord (True) = +1, в то время как для других типов (**Bool, WordBool** и т.д.)

Ord (True) = -1, поэтому такого рода операторы следует использовать с осторожностью! Например, для версии Delphi 6 исполняемый оператор showMessage (' --- ') в следующем цикле *for* не будет выполнен ни разу:

var

L: Bool;

k: Integer;

begin

for L:= False to True do

ShowMessage ('--);

end;

Если заменить тип параметра цикла L в предыдущем примере на **Boolean**, цикл будет работать и сообщение дважды появится на экране. [Для Delphi версии 1 и 2 ord (True) =+1 для любого логического типа.]

*Символьный тип*. Значениями символьного типа является множество всех символов ПК. Каждому символу приписывается целое число в диапазоне 0...255. Это число служит кодом внутреннего представления символа, его возвращает функция ord.

Для кодировки в Windows используется код ANSI (назван по имени American National Standard Institute - американского института стандартизации, предложившего этот код). Первая половина символов ПК с кодами 0... 127 соответствует таблице 1.3. Вторая половина символов с кодами 128...255 меняется для различных шрифтов. Стандартные Windows-шрифты Arial Cyr, Courier New Cyr и Times New Roman для представления символов кириллицы (без букв “ё” и “Ё”) используют последние 64 кода (от 192 до 256): “А”... “Я” кодируются значениями 192..223, “а”... “я” - 224...255. Символы “Ё” и “ё” имеют соответственно коды 168 и 184.

Таблица 1.3 - Кодировка символов в соответствии со стандартом ANSI

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Код | Символ | Код. | Символ | Код. | Символ | Код | Символ |
| 0 | NUL | 32 | BL | 64 | @ | 96 | ' |
| 1 | ЗОН | 33 | ! | 65 | А | 97 | а |
| 2 | STX | 34 | “ | 66 | В | 98 | b |
| 3 | ЕТХ | 35 | # | 67 | С | 99 | с |
| 4 | EOT | 36 | $ | 68 | D | 100 | d |
| 5 | ENQ | 37 | % | 69 | Е | 101 | е |
| 6 | ACK | 38 | & | 70 | F | 102 | f |
| 7 | BEL | 39 | ' | 71 | G | 103 | д |
| 8' | BS | 40 | ( | 72 | Н | 104 | h |
| 9 | HT | 41 | ) | 73 | I | 105 | i |
| 10 | LF | 42 | \* | 74 | J | 106 | j |
| 11 | VT | 43 | + | 75 | К | 107 | k |
| 12 | FF | 44 | F | 76 | L | 108 | 1 |
| 13 | CR | 45 | - | 77 | М | 109 | m |
| 14 | SO | 46 |  | 78 | N | 110 | n |
| 15 | SI | 47 | / | 79 | 0 | 111 | о |
| 16 | DEL | 48 | 0 | 80 | Р | 112 | P |
| 17 | DC1 | 49 | 1 | 81 | Q | 113 | q |
| 18 | DC2 | 50 | 2 | 82 | R | 114 | r |
| 19 | DC3 | 51 | 3 | 83 | S | 115 | s |
| 20 | DC 4 | 52 | 4 | 84 | Т | 116 | t |
| 21 | NAK | 53 | 5 | 85 | U | 117 | u |
| 22 | SYN | 54 | 6 | 86 | V | 118 | v |
| 23 | ETB | 55 | 7 | 87 | W | 119 | W |
| 24 | CAN | 56 | 8 | 88 | х | 120 | x |
| 25 | EM | 57 | 9 | 89 | Y | 121 | У |
| 26 | SUB | 58 | : | 90 | Z | .122 | z |
| 27 | ESC | 59 | ; | 91 | t | 123 | { |
| 28 | FS | 60 | < | 92 | \ | 124 | 1 |
| 29 | GS | 61 | = | 93 | ] | 125 | } |
| 30 | RS | 62 | > | 94 | Л | 126 | ~ |
| 31 | US | 63 | F | 95 |  | 127 | r |

Символы с кодами 0...31 относятся к служебным кодам. Если эти коды используются в символьном тексте программы, они считаются пробелами.

К типу **char** применимы операции отношения, а также встроенные функции:

**Сhаr (в)** - функция типа **char**; преобразует выражение в типа **Byte** в символ и возвращает его своим значением;

**UpCase(CH)** - функция типа **char**; возвращает прописную букву, если сн - строчная латинская буква, в противном случае возвращает сам символ сн (для кириллицы возвращает исходный символ).

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

type

colors = (red, white, blue);

Применение перечисляемых типов делает программы нагляднее.

Соответствие между значениями перечисляемого типа и порядковыми номерами этих значений устанавливается порядком перечисления: первое значение в списке получает порядковый номер 0, второе - 1 и т. д. Максимальная мощность перечисляемого типа составляет 65536 значений, поэтому фактически перечисляемый тип задает некоторое подмножество целого типа **word** и может рассматриваться как компактное объявление сразу группы целочисленных констант со значениями 0, 1 и т. д.

Использование перечисляемых типов повышает надежность программ благодаря возможности контроля тех значений, которые получают соответствующие переменные. В Object Pascal допускается обратное преобразование: любое выражение типа **Word** можно преобразовать в значение перечисляемого типа, если только значение целочисленного выражения не превышает мощности этого типа. Такое преобразование достигается применением автоматически объявляемой функции с именем перечисляемого типа.

*Тип-диапазон*. Тип-диапазон есть подмножество своего базового типа, в качестве которого может выступать любой порядковый тип, кроме типа-диапазона.

Тип-диапазон задается границами своих значений внутри базового типа:

<мин.знач.>..<макс.знач.>

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

Тип-диапазон не обязательно описывать в разделе type, а можно указывать непосредственно при объявлении переменной.

При определении типа-диапазона нужно руководствоваться следующими правилами:

два символа “..” рассматриваются как один символ, поэтому между ними недопустимы пробелы; левая граница диапазона не должна превышать его правую границу.

Тип-диапазон наследует все свойства своего базового типа, но с ограничениями, связанными с его меньшей мощностью. В частности, если определена переменная.

В стандартную библиотеку Object Pascal включены две функции, поддерживающие работу с типами-диапазонами:

*High(х)* - возвращает максимальное значение типа-диапазона, к которому принадлежит переменная х;

*Low (х)* - возвращает минимальное значение типа-диапазона.

**1.1.2 Вещественные типы**

В отличие от порядковых типов, значения которых всегда сопоставляются с рядом целых чисел и, следовательно, представляются в ПК абсолютно точно, значения вещественных типов определяют произвольное число лишь с некоторой конечной точностью, зависящей от внутреннего формата вещественного числа.

Таблица 1.4 - Вещественные типы

|  |  |  |  |
| --- | --- | --- | --- |
| Длина, байт | Название | Количество значащих цифр | Диапазон значений |
| 8  4  8  10  8  8 | Real  Single  Double  Extended  Comp  Currency | 15…16  7…8  15…16  19…20  19…20  19…20 | 5.0\*10e-324…1.7\*10e308  1.5\*10e-45…3.4\*10e38  5.0\*10e324…1.7\*10e308  3.4\*10-4951…1.1\*10e4932  -2e63…+2e63-1  +/-922 337 203 685477,5807 |

В предыдущих версиях Delphi 1...3 тип **Real** занимал 6 байт и имел диапазон значений от 2,9\*10-39 до 1,7\*1038. В версиях 4 и 5 этот тип эквивалентен типу **Double**. Если требуется (в целях совместимости) использовать 6-байтных **Real**, нужно указать директиву компилятора {SREALCOMPATIBILITY ON}.

Как видно из табл. 1.4, вещественное число в Object Pascal занимает от 4 до 10 смежных байт и имеет следующую структуру в памяти ПК.

Здесь s - знаковый разряд числа; е - экспоненциальная часть; содержит двоичный порядок; m - мантисса числа.

*Мантисса* m имеет длину от 23 (для **single**) до 63 (для **Extended**) двоичных разрядов, что и обеспечивает точность 7...8 для **single** и 19...20 для **Extended** десятичных цифр. Десятичная точка (запятая) подразумевается перед левым (старшим) разрядом мантиссы, но при действиях с числом ее положение сдвигается влево или вправо в соответствии с двоичным порядком числа, хранящимся в экспоненциальной части, поэтому действия над вещественными числами называют арифметикой с плавающей точкой (запятой).

Отметим, что арифметический сопроцессор всегда обрабатывает числа в формате **Extended**, а три других вещественных типа в этом случае получаются простым усечением результатов до нужных размеров и применяются в основном для экономии памяти.

Особое положение в Object Pascal занимают типы **comp** и **Currency**, которые трактуются как вещественные числа с дробными частями фиксированной длины: в **comp** дробная часть имеет длину 0 разрядов, т. е. просто отсутствует, в **currency** длина дробной части -4 десятичных разряда. Фактически оба типа определяют большое целое число со знаком, сохраняющее 19...20 значащих десятичных цифр (во внутреннем представлении они занимают 8 смежных байт). В то же время в выражениях **comp** и **currency** полностью совместимы с любыми другими вещественными типами: над ними определены все вещественные операции, они могут использоваться как аргументы математических функций и т. д. Наиболее подходящей областью применения этих типов являются бухгалтерские расчеты.

**1.1.3 Тип дата-время**

Тип дата-время определяется стандартным идентификатором **TDateTime** и предназначен для одновременного хранения и даты, и времени. Во внутреннем представлении он занимает 8 байт и подобно **currency** представляет собой вещественное число с фиксированной дробной частью: в целой части числа хранится дата, в дробной - время. Дата определяется как количество суток, прошедших с 30 декабря 1899 года, а время - как часть суток, прошедших с 0 часов, так что значение 36444,837 соответствует дате 11.10.1999 и времени 20:05. Количество суток может быть и отрицательным, однако значения меньшие -693594 (соответствует дате 00.00.0000 от Рождества Христова) игнорируются функциями преобразования даты к строковому типу.

Над данными типа **TDateTime** определены те же операции, что и над вещественными числами, а в выражениях этого типа могут участвовать константы и переменные целого и вещественного типов.

Поскольку тип **TDateTime** совместим с форматом вещественных чисел, можно без труда определить дату, отстоящую от заданной на сколько-то дней вперед или назад: для этого достаточно соответственно прибавить к заданной дате или отнять от нее нужное целое число.

**1.2 Структурированные типы**

Любой из структурированных типов (а в Object Pascal их четыре: массивы, записи, множества и файлы) характеризуется множественностью образующих этот тип элементов. Каждый элемент, в свою очередь, может принадлежать структурированному типу, что позволяет говорить о возможной вложенности типов. В Object Pascal допускается произвольная глубина вложенности типов, однако суммарная длина любого из них во внутреннем представлении не должна превышать 2 Гбайт [16-разрядные версии операционной системы Windows З.х используют так называемую “сегментную” модель памяти, поэтому в Delphi 1 любой структурированный тип не может занимать более одного сегмента (65536 байт)].

В целях совместимости со стандартным Паскалем в Object Pascal разрешается перед описанием структурированного типа ставить зарезервированное слово *packed*, предписывающее компилятору по возможности экономить память, отводимую под объекты структурированного типа; но компилятор фактически игнорирует это указание: “упаковка” данных в Object Pascal осуществляется автоматачески везде, где это возможно.

**1.2.1 Массивы**

Массивы в Object Pascal во многом схожи с аналогичными типами данных в других языках программирования. Отличительная особенность массивов заключается в том, что все их компоненты суть данные одного типа (возможно, структурированного). Эти компоненты можно легко упорядочить и обеспечить доступ к любому из них простым указанием его порядкового номера.

Описание типа массива задается следующим образом:

<имя типа> = array [ <сп.инд.типов> ] of <тип>;

Здесь <имя типа> - правильный идентификатор; *array, of*- зарезервированные слова {массив, из); <сп.инд.типов> - список из одного или нескольких индексных типов, разделенных запятыми; квадратные скобки, обрамляющие список, - требование синтаксиса; <тип> - любой тип Object Pascal.

В качестве индексных типов в Object Pascal можно использовать любые порядковые типы, имеющие мощность не более 2 Гбайт (т. е. кроме **LongWord** и **Int64**)

Глубина вложенности структурированных типов вообще, а, следовательно, и массивов - произвольная, поэтому количество элементов в списке индексных типов (размерность массива) не ограничено, однако суммарная длина внутреннего представления любого массива не может быть больше 2 Гбайт. В памяти ПК элементы массива следуют друг за другом так, что при переходе от младших адресов к старшим наиболее быстро меняется самый правый индекс массива.

В Object Pascal можно одним оператором присваивания передать все элементы одного массива другому массиву того же типа.

**1.2.2 Записи**

*Запись* - это структура данных, состоящая из фиксированного количества компонентов, называемых полями записи. В отличие от массива компоненты (поля) записи могут быть различного типа. Чтобы можно было ссылаться на тот или иной компонент записи, поля именуются.

Структура объявления типа записи такова:

<имя типа> = record <сп.полей> end;

Здесь <имя типа> - правильный идентификатор; *record/ end* - зарезервированные слова {запись, конец); <сп.полей> - список полей; представляет собой последовательность разделов записи, между которыми ставится точка с запятой.

Каждый раздел записи состоит из одного или нескольких идентификаторов полей, отделяемых друг от друга запятыми.

Предложение *case ... of*, открывающее вариантную часть, внешне похоже на соответствующий оператор выбора, но на самом деле лишь играет роль своеобразного служебного слова, обозначающего начало вариантной части. Именно поэтому в конце вариантной части не следует ставить *end* как пару к *case...of*. (Поскольку вариантная часть - всегда последняя в записи, за ней все же стоит end, но лишь как пара к record). Ключ выбора в предложении *case…of* фактически игнорируется компилятором: единственное требование, предъявляемое к нему в Object Pascal, состоит в том, чтобы ключ определял некоторый стандартный или предварительно объявленный порядковый тип.

Имена полей должны быть уникальными в пределах той записи, где они объявлены, однако, если записи содержат поля-записи, т. е. вложены одна в другую, имена могут повторяться на разных уровнях вложения.

**1.2.3 Множества**

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

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

Описание типа множества имеет вид:

<имя типа> = set of <базовый тип>;

Здесь <имя типа> - правильный идентификатор; *set, of* - зарезервированные слова (множество, из); <базовый тип> - базовый тип элементов множества, в качестве которого может использоваться любой порядковый тип, кроме **Word, Integer, Longint, Int64**.

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

Внутреннее устройство множества таково, что каждому его элементу ставится в соответствие один двоичный разряд (один бит); если элемент включен во множество, соответствующий разряд имеет значение 1, в противном случае - 0. В то же время минимальной единицей памяти является один байт, содержащий 8 бит, поэтому компилятор выделил множествам по одному байту, и в результате мощность каждого из них стала равна 8 элементам. Максимальная мощность множества - 256 элементов. Для таких множеств компилятор выделяет по 16 смежных байт.

И еще один эксперимент: измените диапазон базового типа на 1..256. Хотя мощность этого типа составляет 256 элементов, при попытке компиляции программы компилятор сообщит об ошибке: Sets may have at most 256 elements (Множества могут иметь не более 256 элементов) т. к. нумерация элементов множества начинается с нуля независимо от объявленной в программе нижней границы. Компилятор разрешает использовать в качестве базового типа целочисленный тип-диапазон с минимальной границей 0 и максимальной 255 или любой перечисляемый тип не более чем с 256 элементами (максимальная мощность перечисляемого типа - 65536 элементов).

**1.3 Строки**

Для обработки текстов в Object Pascal используются следующие типы:

короткая строка **shortString** или **string [n]** , где n <= 255;

длинная строка **string**;

широкая строка **WideString**;

нуль-терминальная строка **pchar**.

Общим для этих типов является то, что каждая строка трактуется как одномерный массив символов, количество символов в котором может меняться в работающей программе: для string [n] длина строки меняется от 0 до n, для **string** и **pchar** - от 0 до 2 Гбайт.

В стандартном Паскале используются только короткие строки **String [n]**. В памяти такой строке выделяется n+i байт, первый байт содержит текущую длину строки, а сами символы располагаются, начиная со 2-го по счету байта. Поскольку для длины строки в этом случае отводится один байт, максимальная длина короткой строки не может превышать 255 символов. Для объявления короткой строки максимальной длины предназначен стандартный тип **ShortString** (эквивалент **String[255]**).

В Windows широко используются нуль-терминальные строки, представляющие собой цепочки символов, ограниченные символом #о. Максимальная длина такой строки лимитируется только доступной памятью и может быть очень большой.

В 32-разрядных версиях Delphi введен новый тип **string**, сочетающий в себе удобства обоих типов. При работе с этим типом память выделяется по мере надобности (динамически) и ограничена имеющейся в распоряжении программы доступной памятью.

**1.4 Указатели и динамическая память**

**1.4.1 Динамическая память**

*Динамическая память* - это оперативная память ПК, предоставляемая программе при ее работе. Динамическое размещение данных означает использование динамической памяти непосредственно при работе программы. В отличие от этого статическое размещение осуществляется компилятором Object Pascal в процессе компиляции программы. При динамическом размещении заранее не известны ни тип, ни количество размещаемых данных.

**1.4.2 Указатели**

Оперативная память ПК представляет собой совокупность ячеек для хранения информации - байтов, каждый из которых имеет собственный номер. Эти номера называются адресами, они позволяют обращаться, к любому байту памяти. Object Pascal предоставляет в распоряжение программиста гибкое средство управления динамической памятью - так называемые указатели. Указатель - это переменная, которая в качестве своего значения содержит адрес байта памяти. С помощью указателей можно размещать в динамической памяти любой из известных в Object Pascal типов данных. Лишь некоторые из них (**Byte, Char, ShortInt, Boolean**) занимают во внутреннем представлении один байт, остальные - несколько смежных. Поэтому на самом деле указатель адресует лишь первый байт данных.

Как правило, указатель связывается с некоторым типом данных. Такие указатели будем называть типизированными. Для объявления типизированного указателя используется значок ^, который помещается перед соответствующим типом.

В Object Pascal можно объявлять указатель и не связывать его при этом с каким-либо конкретным типом данных. Для этого служит стандартный тип *pointer*, например:

var

р: Pointer;

Указатели такого рода будем называть нетипизированньти. Поскольку нетипизированные указатели не связаны с конкретным типом, с их помощью удобно динамически размещать данные, структура и тип которых меняются в ходе работы программы.

Как уже говорилось, значениями указателей являются адреса переменных в памяти, поэтому следовало бы ожидать, что значение одного указателя можно передавать другому. На самом деле это не совсем так. В Object Pascal можно передавать значения только между указателями, связанными с одним и тем же типом данных.

**1.4.3 Выделение и освобождение динамической памяти**

Вся динамическая память в Object Pascal рассматривается как сплошной массив байтов, который называется кучей.

Память под любую динамически размещаемую переменную выделяется процедурой New. Параметром обращения к этой процедуре является типизированный указатель. В результате обращения указатель приобретает значение, соответствующее адресу, начиная с которого можно разместить данные. Значение, на которое указывает указатель, т. е. собственно данные, размещенные в куче, обозначаются значком ^, который ставится сразу за указателем. Если за указателем нет значка ^, то имеется в виду адрес, по которому размещены данные. Имеет смысл еще раз задуматься над только что сказанным: значением любого указателя является адрес, а чтобы указать, что речь идет не об адресе, а о тех данных, которые размещены по этому адресу, за указателем ставится ^ (иногда об этом говорят как о разыменовании указателя).

Динамически размещенные данные можно использовать в любом месте программы, где это допустимо для констант и переменных соответствующего типа

Динамическую память можно не только забирать из кучи, но и возвращать обратно. Для этого используется процедура Dispose. Например, операторы

*Dispose(pJ);*

*Dispose(pR);*

вернут в кучу память, которая ранее была закреплена за указателями pJ и pR (см. выше).

Замечу, что процедура Dispose (pPtr) не изменяет значения указателя pPtr, а лишь возвращает в кучу память, ранее связанную с этим указателем. Однако повторное применение процедуры к свободному указателю приведет к возникновению ошибки периода исполнения. Освободившийся указатель программист может пометить зарезервированным словом nil.

**1.5 Псевдонимы типов**

Для любого типа можно объявить сколько угодно псевдонимов. Например:

type

TMyInteger = Integer;

В дальнейшем псевдоним можно использовать так же, как и базовый тип:

var

Mylnt: TMyInteger;

begin

Mylnt := 2\*Round(pi);

end;

Такого рода псевдонимы обычно используются для повышения наглядности кода программы. Однако в Object Pascal можно объявлять строго типизированные псевдонимы добавлением зарезервированного слова type перед именем базового типа:

type

TMyIntegerType = type Integer;

var

MylntVar: TMyIntegerType;

С точки зрения компилятора, типизированные псевдонимы совместимы с базовым типом в различного рода выражениях, но фактически они объявляют новый тип данных, поэтому их нельзя использовать в качестве формальных параметров обращения к подпрограммам вместо базового типа. Если, например, объявлена процедура

function MylntFunc(APar: integer): Integer;

begin

end;

то такое обращение к ней

MylntFunc(MylntVar)

будет расценено компилятором как ошибочное.

Строго типизированные псевдонимы заставляют компилятор вырабатывать информацию о типе для этапа прогона программы (RTTI - Run-Time Type Information). Эта информация обычно используется средой Delphi для обеспечения функционирования разного рода редакторов.