ListBox, CheckListBox, ValueListEditor, ComboBox и ComboBoxEx - выбор из списка

1

[http://cubook.supernew.org/plugins/content/egoltlike/assets/images/like.png](javascript:void(null))

-3

[http://cubook.supernew.org/plugins/content/egoltlike/assets/images/dislike.png](javascript:void(null))

4

**Компоненты ListBox и [CheckListBox](http://cubook.supernew.org/object/231-checkbox-i-checklistbox-indikatory.html)**

Компоненты **ListBox** и **CheckListBox** отображают списки строк и позволяют пользователю выбрать в них нужную строку. Основное свойство обоих компонентов, содержащее список строк *Items*, имеющий тип*TStrings*. Заполнить его во время проектирования можно, нажав кнопку с многоточием около этого свойства в окне Инспектора Объектов. Во время выполнения работать с этим свойством можно, пользуясь свойствами и методами класса *TStrings* *Clear*, *Add* и другими. Этот же класс позволяет поставить в соответствие каждой строке некоторый объект. Тогда выбор пользователем строки в списке можно программно соотносить с этим объектом.

В компоненте **ListBox** имеется свойство *MultiSelect*, разрешающее пользователю множественный выбор в списке, то пользователь может выбрать только один элемент списка. В этом случае можно узнать индекс выбранной строки из свойства *ItemIndex*, доступного только во время выполнения. Если ни одна строка не выбрана, то ItemIndex = -1.

Например, следующий код проверяет выбор пользователя. Если выбор не сделан, появляется сообщение «Вы не сделали свой выбор». Если один из элементов списка выбран, то появляется сообщение вида «Ваш выбор ...: ...», где вместо первого многоточия отображается номер выбранной строки, а вместо второго многоточия - текст выбранной строки:

if(ListBox1->ItemIndex < 0)  
ShowMessage("Вы не сделали свой выбор");  
else ShowMessage("Ваш выбор " +  
IntToStr(ListBox1->ItemIndex + 1) + ": " + ListBox1->Items->Strings[ListBox1->ItemIndex]);

Начальное значение *ItemIndex* невозможно задать во время проектирования. По умолчанию ItemIndex = -1. Это означает, что ни один элемент списка не выбран. Если вы хотите задать этому свойству какое-то другое значение, т.е. установить выбор по умолчанию, который будет показан в момент начала работы приложения, то сделать это можно, например, в обработчике события *OnCreate* формы, введя в него оператор вида:

ListBox1->ItemIndex = 0;

Если допускается множественный выбор (MultiSelect = true), то значение *ItemIndex* соответствует тому элементу списка, который находится в фокусе. При множественном выборе можно проверить, выбран ли данный элемент, по свойству *Selected[int Index]* типа **bool**. Например, следующий код отображает сообщения вида "Выбрана строка ...: ..." (похожие на предыдущий пример) обо всех выбранных строках:

for(int i=0; i < ListBox1->Items->Count; i++)  
if(ListBox1->Selected[i])  
ShowMessage("Выбрана строка " + IntToStr (i+1) + ": " + ListBox1->Items->Strings[i]):

На способ множественного выбора при MultiSelect = true влияет еще свойство *ExtendcdSelect*. Если ExtendedSelect = true, то пользователь может выделить интервал элементов, выделив один из них, затем нажав клавишу *Shift* и переведя курсор к другому элементу. Выделить не прилегающие друг к другу элементы пользователь может, если будет удерживать во время выбора нажатой клавишу *Ctrl*. Если же ExtendedSelect = false, то клавиши *Shift* и *Ctrl* при выборе не работают.

Свойство *Columns* определяет число столбцов, в которых будет отображаться список, если он не помещается целиком в окне компонента *ListBox*.  
Свойство *Sorted* позволяет упорядочить список по алфавиту. При Sorted = true новые строки в список добавляются не в конец, а по алфавиту.  
Свойство *AutoComplete*, если установить его в true, позволяет пользователю быстро находить строку списка, нажимая клавишу, соответствующую ее первому символу.  
Свойство *Style*, установленное в *lbStandard* (значение по умолчанию) соответствует списку строк. Другие значения *Style* позволяют отображать в списке не только текст, но и изображения, а в C++Builder 6 введена возможность создавать виртуальные списки.

Значения *IbOwnerDrawFixed* и *lbOwnerDrawVariable* используются для занесения в список изображений. При этом *IbOwnerDrawFixed* означает, что высота всех элементов списка одинакова, а значение *lbOwnerDrawVariable* означает, что высота элементов может быть различной. Значения*lbVirtual* и *lbVirtualOwnerDraw* соответствуют виртуальным спискам соответственно со строками текстов и изображений.

При значении *Style*, равном *IbOwnerDrawFixed* или *lbOwnerDrawVariable*, в момент, когда должна рисоваться какая-то строка списка, наступает событие *OnDrawItem*.

Параметр *Control* является указателем на список, в котором происходит событие.  
Параметр *Index* указывает индекс элемента, который должен быть перерисован.  
Параметр *Rect* типа *TRect* указывает область канвы списка, соответствующую рисуемому элементу списка.  
Параметр *TOwnerDrawState* является множеством, элементами которого могут быть значения*odSelected* - строка выделена, *odFocused* строка находится в фокусе и ряд других.  
В обработчике события *OnDrawItem* надо методами работы на канве нарисовать изображение элемента.

При значении *Style* равном *lbOwnerDrawFixed* перед прорисовкой наступает только событие*OnDrawItem*. При *Style = lbOwnerDrawVariable* перед этим событием наступает другое -*OnMeasureltem*, в котором надо указать высоту элемента.

Параметры *Control* и *Index* имеют тот же смысл, что и в обработчике *OnDrawItem*, а значение параметра*Height* надо задать равным высоте данного элемента списка.  
Событие *OnData* наступает в тот момент, когда приложению надо отобразить очередную строку списка.  
Параметр *Control* - виртуальный список, в котором происходит событие. Параметр *Index* - это индекс строки, которая должна отображаться. А в параметр *Data* надо задать отображаемый текст. Например, оператор занесет в строку текст «Строка ...», в котором вместо многоточия будет номер строки:

Data = "Строка " + IntToStr(Index + 1);

Параметр *Index* указывает индекс строки, а в параметр *DataObject* заносится связываемый со строкой объект. Параметр *FindString* - это искомая строка, индекс которой надо вернуть оператором *return*.

**Компонент CheckListBox**

Имеется еще один компонент, очень похожий на **ListBox** - это список с индикаторами **CheckListBox**. Выглядит он так же, как **ListBox**, но около каждой строки имеется индикатор, который пользователь может переключать. Индикаторы можно переключать и программно, если список используется для вывода данных и необходимо в нем отметить какую-то характеристику каждого объекта, например, наличие товара данного наименования на складе.

В C++Builder 6 в **CheckListBox** появились новые свойства, позволяющие зрительно разбить список на несколько разделов с помощью заголовков. Свойство *Header* представляет собой индексированный массив булевых значений, определяющих, является ли соответствующая строка заголовком (значение true), или это обычная строка с индикатором (значение false). Свойство *Header* - только времени выполнения и должно заполняться программно (по умолчанию все значения равны false).  
Например, операторы задают в качестве заголовков вторую и десятую строки:

CheckListBox1->Header[1] = true;  
CheckListBox1->Header[9] = true;

Заголовки отображаются в строках с цветом фона, определяемым свойством *HeaderBackgroundColor*, и цветом надписи, задаваемым свойством *HeaderColor*.

**Компонент ValueListEditor**

Компонент **ValueListEditor** - это окно редактирования списка строк вида «имя = значение». Окно имеет две колонки с заголовками «Key» для имен и «Value» для значений. Заголовки можно изменить во время проектирования или программно, используя свойство *TitleCaptions* типа *TStrings*. Первая строка этого свойства соответствует первой колонке (именам), вторая - второй колонке.

Свойство *Strings* типа *TStrings* содержит список всех строк. Во время проектирования он может быть заполнен редактором строк, вызываемым из Инспектора Объектов. Во время выполнения пользователь может заполнять его, если в компоненте разрешено редактирование. Можно также программно присвоить свойству *Strings* содержание некоторого другого списка типа *TStrings*. Все строки присваиваемого списка должны иметь вид «имя = значение». Тогда содержимое списка отобразится в колонках компонента. Методы класса *TStrings* позволяют манипулировать списком, добавлять строки, удалять и т.п.  
Свойство *Keys* является индексированным списком имен. Индексы начинаются с 1.  
Свойство *[AnsiString](http://cubook.supernew.org/manual-c/types/197-ansistring.html) Values[const AnsiString Key]* позволяет задать или прочитать значение из строки с именем *Key*. Если свойство используется для задания значения, а имени *Key* в списке нет, то в список добавляется новая строка с указанным именем и значением. Если свойство используется для чтения, а имени Key в списке нет, то возвращается пустая строка. Доступ к именам и значениям дают также свойства Strings->Names и Strings->Values (как в обычных списках типа TStrings).  
 Еще один способ доступа к элементам строк дает свойство *AnsiString Cells[int ACol][int ARow]*. В этом свойстве *ACol* = 0 соответствует колонке имен, a *ACol* = 1 - колонке значений. *ARow* = 0 соответствует строке заголовков, a *ARow* = 0 - строкам списка.  
Свойство только для чтения *RowCount* показывает число строк, включая строку заголовка.  
Свойство *KeyOptions* позволяет определить операции, доступные пользователю при редактировании колонки имен. Это свойство является множеством, пустым или содержащим элементы *keyEdit, keyAdd, keyDelete, keyUnique*.

Элемент *keyEdit* позволяет пользователю редактировать имена в первой колонке.  
Элемент *keyAdd* позволяет пользователю добавлять в список новые строки, используя клавишу *Insert* или перемещая курсор клавишей со стрелкой ниже позиции последней строки.  
Элемент *keyAdd* можно включать только вместе с *keyEdit*.  
Элемент *keyDelete* позволяет пользователю удалять выделенную строку клавишей *Delete*.  
Элемент *keyUnique* не позволяет пользователю добавить новую строку с именем, уже имеющимся в списке. Такая попытка приводит к генерации исключения.

Свойство только времени выполнения *ItemProps* позволяет управлять способом редактирования пользователем значения каждого из элементов списка. Элемент задается или его именем, или индексом строки (начинаются с 0). Массив *ItemProps* содержит объекты класса *TItemProp*, описывающие доступ к соответствующему элементу. Свойства этого класса задают способы редактирования значений:

|  |  |  |
| --- | --- | --- |
| EditMask | Маскированный ввод. | |
| EditStyle | Стиль редактирования: | |
|  | esSimple | Обычное окно редактирования. |
|  | esEllipsis | Кнопка с многоточием, при нажатии на которую возникает событие OnEditButtonClick. В обработчике этого события можно предусмотреть вызов какого-то окна специализированного редактора свойства. |
|  | esPickList | Выпадающий список, строки которого задаются свойством PickList или в обработчике события OnGetPickList (можно сочетать эти две возможности, задавая список в PickList и модифицируя его в обработчике OnGetPickList). |
| KeyDesc | Имя, появляющееся в строке данного элемента списка в колонке «Кеу». | |
| MaxLength | Максимальное число вводимых символов. | |
| PickList | Список типа TStrings, появляющийся в выпадающем списке при значении EditStyle = esPickList. | |
| ReadOnly | Только для чтения, редактирование запрещено. Например, сочетание ReadOnly = true и стиля EditStyle, равного esPickList или esEllipsis, не даст пользователю задать значение иначе, как выбрав его из списка или введя с помощью редактора свойства. | |

Стиль EditStyle = esPickList приводит к тому, что при попытке пользователя редактировать значение он видит кнопку выпадающего списка, из которого он может выбрать соответствующее значение. Если при этом задать ReadOnly = true, то пользователь не сможет изменить значение иначе, чем выбором из списка. В качестве примера приведем операторы, обеспечивающие выбор:

Valedit::TItemProp \*ItemProp = ValueListEditor1->ItemProps["Отдел"];  
ItemProp->EditStyle = esPickList;  
ItemProp->PickList->Add("Управление");  
ItemProp->PickList->Add("Цех 1");  
ItemProp->PickList->Add("Цех 2") ;  
ItemProp->ReadOnly = true;

Стиль EditStyle = esEllipsis приводит к тому, что при попытке пользователя редактировать значение он видит кнопку с многоточием. При нажатии на нее возникает событие *OnEditButtonClick*. В его обработчике можно, например, вызвать какое-то диалоговое окно, в котором пользователь должен ответить на какие-то вопросы. В результате будет выработано значение, которое занесется в строку списка.  
Например:

ValueListEditorl->Values["Должность"] = "Дворник";

Если к тому же задать для этой строки ReadOnly = true, то пользователь не сможет задать значение иначе, чем в результате работы этого диалога.

Свойство *DisplayOptions* является множеством, которое может содержать элементы *doColumnTitles* - строка заголовка фиксирована (не прокручивается), *doKeyColFixed* - ширина колонки имен неизменна,*doAutoColResize* - при изменении ширины компонента ширина колонок автоматически изменяется.  
Свойство *Options* содержит множество опций, управляющих отображением линий таблицы, возможность для пользователя изменять размеры колонок и т.п.

**Компонент ComboBox**

Стиль изображения компонента **ComboBox** определяется его свойством *Style*, которое может принимать следующие основные значения:

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| csDropDown | Выпадающий список со строками одинаковой высоты и с окном редактирования, позволяющим пользователю вводить или редактировать текст. |
| csSimple | Развернутый список со строками одинаковой высоты и с окном редактирования, позволяющим пользователю вводить или редактировать текст. При таком стиле надо делать во время проектирования или программно высоту списка такой, чтобы в нем могло отобразиться требуемое число строк. |
| csDropDownList | Выпадающий список со строками одинаковой высоты, не содержащий окна редактирования. |
| csOwnerDrawFixed | Выпадающий список со строками одинаковой высоты, в которых могут отображаться изображения и текст. |
| csOwnerDrawVariable | Выпадающий список со строками разной высоты, в которых могут отображаться изображения и текст. |

При стилях *csOwnerDrawFixed* и *csOwnerDrawVariable* изображения рисуются на канве в обработчике события *OnDrawItem* так же, как это рассказывалось ранее для **ListBox**. При стиле*csOwnerDrawVariable* перед рисованием возникает событие *OnMeasureItem*, в обработчике которого надо задать высоту элемента.

Выбор пользователя или введенный им текст можно определить по значению свойства *Text*. Если же надо определить индекс выбранного пользователем элемента списка, то можно воспользоваться обсуждавшимся в компоненте **ListBox** свойством *ItemIndex*. Причем для **ComboBox** задание начального значения *ItemIndex* еще актуальнее, чем для **ListBox**.

Если начальное значение не задано, то в момент запуска приложения пользователь не увидит в окне компонента одно из возможных значений списка и, вероятнее всего, не очень поймет, что с этим окном надо делать.

Если в окне проводилось редактирование данных, то *ItemIndex* = -1. По этому признаку можно определить, что редактирование проводилось.

Свойство *MaxLength* определяет максимальное число символов, которые пользователь может ввести в окно редактирования. Если *MaxLength* = 0, то число вводимых символов не ограничено.

Свойство *DropDownCount* указывает число строк, появляющихся в выпадающем списке без возникновения полосы прокрутки.

Как и в компоненте **ListBox**, свойство *Sorted* позволяет упорядочить список по алфавиту. При Sorted = true новые строки в список добавляются не в конец, а по алфавиту.

**Компоненте ComboBoxEx**

Компоненте **ComboBoxEx** во многом подобен **ComboBox**. Различие, прежде всего, заключается в том, что в *ComboBoxEx* легче, чем в **ComboBox**, вводить изображения в элементы списка. С другой стороны, некоторые возможности **ComboBox**, например, возможность сортировки, в **ComboBoxEx** не поддерживаются. Невозможно также в этом списке рисование на канве.

Изображения, отображаемые в элементах, должны содержаться в компоненте **ImageList**. Ссылка на этот компонент задается в свойстве *Images*. Ввод элементов списка во время проектирования осуществляется редактором коллекций, вызываемым шелчком на кнопке с многоточием в окне Инспектора Объектов около свойства *ItemsEx*. Перед вами откроется окно. Кнопка Add New (левая) позволяет ввести новый элемент. Если вы ввели элемент и выделили его, то в окне Инспектора Объектов увидите его свойства. Основные из них: *Caption* - текстовая строка, *ImageIndex* - индекс соответствующей пиктограммы в списке *ImageList*, *Indent* - отступ от левого поля элемента списка.

Свойство *Style* определяет стиль элемента и может быть равным *csExDropDown*, *csExSimple* или*csExDropDownList*. Смысл этих значений подобен рассмотренному выше для компонента **ComboBox**.

**Типы данных в DELPHI**

    С помощью типов данных программист указывает компилятору, как хранить информацию в программе. При объявлении переменной необходимо указать ее тип. Одни типы уже определены в языке, другие программисту приходится задавать самому. В ранних языках программирования допускалось ограниченное число типов данных, и Pascal оказался одним из первых языков, допускающих определение в программе новых типов.  
Типы данных, определяемые пользователем, обычно задаются в разделе определения типов программы или модуля (unit), однако это можно делать и внутри процедур или функции. Объявления типов действуют в пределах того блока, в котором они размещены. Вне этого блока ссылаться на такие типы нельзя. Внутри же они заменяют все внешние типы с тем же именем. Объявленные типы данных можно применять в любом месте области их видимости; запрещена только ссылка определяемого типа на самого себя (тут, однако, есть одно исключение, касающееся указателей).  
    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 объявлений 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 компонентов программы. Выглядят же они следующим образом.

type  
    typel = type definitioni;              //Новые типы данных определяются в разделе "type". Каждому новому  
                                                    // типу присваивается имя, затем он определяется через уже  
                                                    //существующие типы.  
    type2 = type\_\_definition2;        // В одном разделе "type" можно объявить несколько типов.  
                                                    //Самое простое определение типа состоит из имени типа,  
    type3 = typel;                           // определенного ранее.  
                                                    // Новые переменные объявляются в  
    var                                           // разделе "var". Каждой новой  
        var1: type definitions;            // переменной сначала присваивается имя, а затем — тип (на основе  
                                                    // ранее определенных типов).  
        var2, var3: type definition4;   // В одном разделе "var" можно объявить несколько переменных.  
                                                    // Нескольким переменным можно присваивать один и тот же тип.  
        var4 : typel;                          // Программу легче читать, если переменным присвоены  
                                                    //существующие типы.

    Синтаксис Object Pascal позволяет одновременно конструировать исключительно сложные типы и определение переменных. Однако определение типов в разделах type тех или иных блоков 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 функций, в которых имя типа может использоваться как часть выполняемого оператора. Например, функция SizeOf (Т) возвращает количество байтов, занимаемых переменной Т.  
    Функция SizeOf очень важна для написания эффективных программ. Многие из определенных в Object Pascal типов имеют очень сложную структуру и могут занимать в памяти довольно много места. При этом элементы таких типов созданы скорее для представления значений в некотором логическом порядке, а не для того, чтобы занимать место в памяти. Функция SizeOf избавляет программиста от необходимости вычислять объем данных в подобных случаях.

***Простые типы данных***

**Порядковые типы**  
**Целые типы**  
**Символьные типы**  
**Булевы типы**  
**Перечислимые типы**  
**Поддиапазонные типы**  
**Действительные типы**

    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 эти простые типы данных разбиты на две группы: порядковые, представляющие данные разных объемов, которыми процессор может легко манипулировать, и действительные, представляющие приближенно математические действительные числа. Разделение типов на порядковые и действительные несколько условно. Точно так же простые 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 — самые простые. В этих типах информация представляется в виде отдельных элементов. Связь между отдельными элементами и их представлением в памяти определяет естественные отношения порядка между этими элементами. Отсюда и название порядковые.  
    В Object Pascal определены три группы порядковых типов и два типа, определяемых пользователем. Группы — это целые, символьные и булевы типы. Порядковые типы, задаваемые пользователем, — это перечисления и поддиапазоны.  
    Все значения любого порядкового типа образуют упорядоченную последовательность, и значение переменной порядкового типа определяется его местом в этой последовательности. За исключением переменных целых типов, значения которых могут быть как положительными, так и отрицательными, первый элемент любого порядкового типа имеет номер 0, второй элемент — номер 1 и т.д. Порядковый номер целого значения равен самому значению. Отношение порядка определяет общие для данных всех порядковых типов операции. Некоторые стандартные функции такого вида встроены в Object Pascal. Они представлены в табл. 1.1.

    Для всех порядковых типов в Object Pascal существует операция задания типа для преобразования целых значений в значения соответствующих порядковых типов. Если Т — имя порядкового типа, а Х — целое выражение, то Т (X) воз-вращает значение Т с порядковым номером X.

**Таблица 1.1.** Операции над порядковыми типами

|  |  |
| --- | --- |
| **Операция** | **Описание** |
| Low (T) | Минимальное значение порядкового типа Т |
| High(T) | Максимальное значение порядкового типа Т |
| Ord(X) | Порядковый номер значения выражения порядкового типа. Для целого выражения — просто его значение. Для остальных порядковых типов Ord возвращает физическое 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 Х-1 |
| Succ(X) | Следующее по порядку значение. Для целых выражений эквивалентно Х+1 |
| Dec(V) | Уменьшает значение переменной на 1. Эквивалентно V := Pred(V) |
| Inc(V) | Увеличивает значение переменной на 1. Эквивалентно V := Succ(V) |

**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 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, a 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 табл. 1.2.

**Таблица 1.2.**Физические целые типы

|  |  |  |
| --- | --- | --- |
| **Тип** | **Диапазон значении** | **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 табл. 1.3, они никак не связаны с диапазонами соответствующих физических типов. Обратите внимание, что в Delphi по умолчанию задано 32-разрядное представление.

**Таблица 1.3.**Логические целые типы

|  |  |  |
| --- | --- | --- |
| Тип | Диапазон значений | Физический формат |
| Integer | -32 768-32 767 | 16 бит, со знаком (SmalIInt) |
| 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 147483647 | 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 Х |
| Х Div Y | Возвращает целую часть частного деления Х на Y |
| Х Mod Y | Возвращает остаток частного деления Х на Y |
| Odd (X) | Возвращает булево True (истина), если Х — нечетное целое, и False (ложь) — в противном случае |
| Sqr (X) | Возвращает целый квадрат Х (т.е. Х\*Х) |

**Символьные типы**  
    Смысл символьных данных 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 том же компьютере мог использовать каждый.  
    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. Дополнительный бит удваивает число возможных представлений символов, но реализации расширенного набора символов ASCII часто бывают далеки от стандарта. В данной версии Delphi определен набор 8-битовых символов, известный как расширенный (extended) ANSI (American National Standards Institute — Американский национальный институт стандартов). Как бы то ни было, символьную схему приходится воспринимать так, как ее воспринимает операционная система. Для оконных операционных систем фирмы Microsoft это схема ANSI, включающая ограниченное число предназначенных для вывода международных знаков. В стремлении же 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 и один логический.  
Физические типы перечислены ниже.

|  |  |
| --- | --- |
| AnsiChar | Однобайтовые символы, упорядоченные в соответствии с расширенным набором символов ANSI |
| WideChar | Символы объемом в слово, упорядоченные в соответствии с международным набором символов UNICODE. Первые 256 символов совпадают с символами ANSI |

     Символьные типы объемом в двойное слово (32 бит) отсутствуют.  
    Логический символьный тип именуется char. В классическом языке Pascal char— единственный символьный тип. В Delphi char всегда соответствует физическому типу данных AnsiChar. У американских программистов ассоциация символа с однобайтовой ячейкой памяти укоренилась за долгие годы настолько, что им зачастую просто не приходит в голову, что можно использовать другие схемы кодирования. Однако дискуссии по интернационализации программ в Internet и World Wide Web могут существенно изменить их отношение к проблеме объема символьных данных. Применяя логический тип char, следует делать реализации для других микропроцессоров и операционных систем, в которых char может определяться как WideChar. При написании программ, которые могут обрабатывать строки любого размера, для указания этого размера рекомендуется применять функцию SizeOf, не задавая ее жестко постоянной. Функция Ord (С), где С — любая переменная символьного типа, возвращает целое значение, которым символ С представлен в памяти.

|  |  |
| --- | --- |
| Chr (X) | Преобразует целую переменную в переменную типа char с тем же порядковым номером. В Delphi это эквивалентно заданию типа Char (X) |
| UpCase | Преобразует 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 одно из двух. Бит может обозначать 0 или 1, ДА или НЕТ, ВКЛЮЧЕНО или ВЫКЛЮЧЕНО, ВЕРХ или НИЗ, СТОЯТЬ или ИДТИ. В Object Pascal информация о чем-либо, что можно представить как ИСТИНА (True) или ЛОЖЬ (False), хранится в переменных булевых типов. Всего таких типов че-тыре, и они представлены в табл. 1.4.

**Таблица 1.4.**Размеры переменных булевых типов

|  |  |
| --- | --- |
| 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, 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**  
    Type enum type = (first value, value2, value3, last value);  
    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 следующим образом:  
    Type Boolean = (False, True);  
    С помощью типа Boolean в Object Pascal выполняются сравнения, большинство же 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). Это делается совсем просто:  
    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 три идентификатора— это значения типа MyColor. Подобно символьным и булевым типам перечислимые не являются числами, и использовать их наподобие чисел не имеет смысла. Однако перечислимые типы относятся к порядковым, 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**  
    Переменные поддиапазонного типа содержат информацию, соответствующую некоторому pаданному диапазону значений исходного типа, представляющего любой порядковый тип, кроме поддиапазонного. Синтаксис определения поддиапазонного типа имеет следующий вид:

    Type subrange type = low value...high value;

    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. Действительные типы описываются в табл. 1.5.

**Таблица 1.5.** Действительные типы.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Тип** | **Порог** | **Максимальное значение** | **Количество значащих цифр** | **Объем (байт)** |
| Real | 2.9E-39 | 1.7Е38 | 11-12 | 6 |
| Single | 1.5E-45 | 3.4Е38 | 7-8 | 4 |
| Double | 5.0E-324 | 1.7Е308 | 15-16 | 8 |
| Extended | 3.4E-4932 | 1.IE4932 | 19-20 | 10 |
| Comp | 1.0 | 9.2Е18 | 19-20 | 8 |
| Currency | 0.0001 | 9.2Е14 | 19-20 | 8 |

    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.  
    Заметьте, именно почти. 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 Engi-neers — 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 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 определены два формата с фиксированной запятой. Тип Comp (computational — вычислительный) содержит только целые числа в диапазоне от -263+1 до 263-1, что примерно соответствует диапазону от —9,2х1018 до 9,2х1018. При программировании операций с американской валютой разработчикам обычно приходится искать естественный способ записи денежных сумм, в котором целая часть числа определяет количество долларов, дробная — центов. Если такие значения записывать в переменные типа Comp, придется представлять их в виде целого числа центов. В этом случае следует умножать значение на 100 для обращения центов в доллары, а затем делить на 100, чтобы снова получить центы.  
    Этих забот можно избежать, если воспользоваться типом Currency. В этом случае задачу выбора масштаба возьмет на себя компилятор. 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 (не на 100!) 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 Delphi есть модуль System, содержащий ряд процедур обработки данных действительных типов. Наиболее распространенные из них перечислены в табл. 1.6. Много полезных процедур содержится также в модулях SysUtils и Math.

**Таблица 1.6.** Функции действительных типов

|  |  |
| --- | --- |
| Функция | Возвращаемое значение |
| Abs (x) | Абсолютная величина х |
| АгсТаn(х) | Арктангенс х |
| Cos (х) | Косинус х (х выражается в радианах, а не в градусах) |
| Ехр (х) | Экспоненциальная функция от х |
| Frac(x) | Дробная часть х |
| Int (х) | Целая часть х. Несмотря на название, возвращает действительное значение (с плавающей запятой), т.е. просто устанавливает нуль в дробной части |
| Ln (х) | Натуральный логарифм от х |
| Pi | Число Пи (3.1416...) |
| Round (х) | Ближайшее к х целое значение. Возвращает значение целого типа. Условие "ближайшее к х" не работает, если верхнее и нижнее значения оказываются равноудаленными (например, ес-ли дробная часть точно равна 0,5). В этих случаях Delphi перекладывает решение на опера-ционную систему. Обычно процессоры Intel решают эту задачу в соответствии с рекоменда-цией IEEE округлять в сторону ближайшего четного целого числа. Иногда такой подход на-зывают "банкирским округлением" |
| Sin(x) | Синус х |
| Sqr(x) | Квадрат х, т.е. X\*X |
| Sqrt (х) | Квадратный корень от х |
| Тrunc (х) | Целая часть х. В отличие от Int, возвращающей действительное значение, Trunc возвращает целое |

***Строковые типы***

    В выражениях Delphi поддерживает три физических строковых формата: короткий (ShortString), длинный (LongString) и широкий (WideString). Их можно комбинировать в операторах присваивания и выражениях (все необходимые преобразования Delphi выполняет автоматически).  
    Переменные типов AnsiString и WideString — это динамически распределяемые массивы символов, максимальная длина которых ограничивается только наличием 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. Первый его элемент задает динамическую длину строки, которая может принимать значения от 0 до 255 символов. Символы, составляющие строку, занимают места 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 $Н. По умолчанию задается { $Н+}, и String совпадает с AnsiString. Если задать команду {$Н- }, 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 стооках обозначается нулем.  
    Физически строки с нуль-окончанием подобны массивам символов с нумерацией элементов от нуля, наподобие array [ 0 . . X] of char, где Х — некоторое положительное целое, большее нуля, хотя никаких объявлении подобного рода не происходит. Вместо этого определяется переменная-указатель PChar и распределяется необходимый объем памяти. При необходимости строке AnsiString можно присвоить тип PChar.  
В табл. 1.7 перечислены некоторые процедуры и функции обработки данных строковых типов.

**Таблица 1.7.** Строковые функции

|  |  |
| --- | --- |
| **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, tar-get, pos) | Вставляет строку source в строковую переменную target, начиная с позиции pos |
| Length (s) | Возвращает динамическую длину строки. Подобна функциям LEN в Basic и strlen — в C/C++ |
| Pos(substring, s) | Возвращает место первого вхождения подстроки substring в строку s. Подобна функциям SUBSTR в Basic и strstr () — в C/C++ |
| SetLength(s, newlen) | Задает новую динамическую длину newlen строковой переменной s |
| SetString | Задает содержимое и длину строки |
| Str(x, s) | Преобразует численное значение х 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***

**Записи**  
**Фиксированные записи**  
**Вариантные записи**  
**Массивы**  
**Множества**  
**Файловый тип**

    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 дополнения существующих типов.

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

    record  
        fieldnamel: fieldtypel;  
        fieldname2, fieldname3: fieldtype2;  
        case optional tagfield: required ordinal type of  
        1: variantnamel: 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  
        fieldnamel: fieldtypel;  
        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.Fieldnamel

    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: variantnamel: varianttype3;  
        2, 3: variantname2: varianttype4;  
    end;

    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.  
    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 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.

    array [ordinal\_type] of type\_definition;  
    array [ordinal typel, 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.  
    Количество элементов массива равно произведению количеств элементов во всех измерениях.  
Для обращения к элементу массива укажите имя этого массива и индекс элемента в квадратных скобках. Пусть, например, массив определен следующим образом:

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

    Тогда обращение к его третьему элементу будет выглядеть, как MyArray[З], 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.  
    var MyAlphaSet: CharSet;           // Переменная типа CharSet.

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

    MyAlphaSet := ['А', 'Е', 'Г, 'О', 'U', 'Y']; // Все прописные гласные.

    Пустые квадратные скобки задают пустое множество, не содержащее ни одного элемента. Это относится ко всем множественными типам.

**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 Typel  // Файл определенного типа, содержащий  
                        // записи фиксированной длины.  
    file                // Файл без типа или "блочный".  
    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. Их реализация должна оставаться чем-то наподобие черной магии. В Basic файлы обозначаются числовыми значениями — дескрипторами. В C/C++ программисты манипулируют указателями на структуру FILE. И только в Delphi файловая структура — это переменная.

***Указательные типы***

    Переменная указательного типа содержит значение, указывающее на переменную обычного типа — адрес этой переменной (табл. 1.8).

    pointer   // Указатель без типа.  
    ^typel    // Указатель с типом.

    Если исходный тип (тип переменной, на которую 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.

**Таблица 1.8.**Средства работы с указателями

|  |  |
| --- | --- |
| 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 |

**Указатели и адресные функции**  
    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.

***Процедурные типы***

**Тип Variant**  
**Вариантные значения**  
**Процедуры обработки вариантных массивов**  
**OLE Automation**

    Указатели удобны для записи информации заданного типа и обращения к переменным. Однако записывать адреса процедур и функций довольно опасно. Программистам, работавшим с ранней версией Turbo Pascal, приходилось основательно изучать способы размещения процедур и функций в памяти 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.  
    В Delphi указатели процедурного типа на методы применяются для сопоставления событий с образцами текста программы. С точки зрения синтаксиса, единственное отличие процедурного типа для метода от обычного процедурного типа состоит в фразе of object, следующей за прототипом процедуры или функции в случае метода. Особым образом применяется в процедурных методах указательное значение Nil. Это единственное указательное значение, которое можно присвоить процедурной переменной. После присвоения такого значения процедурная переменная становится неопределенной. Состояние определенности можно проверить с помощью функции Assigned.  
    Глобальные процедурные типы и процедурные типы для методов взаимно несовместимы. Нельзя присваивать значение одного типа переменной другого.  
    Физически процедурные типы в Delphi совпадают с указательными, однако они различаются синтаксически, поэтому нельзя обращаться к функции или процедуре через указатель. Тем не менее при обращении к процедурной переменной задействуется именно значение указательного типа. В C/C++ переменная может иметь тип указателя на функцию. В версиях языка С до введения стандарта ANSI для обращения к соответствующей функции приходилось явно адресовать указатель. В версиях С, со-ответствующих стандартам ANSI, возможны как явная, так и неявная адресации указателей. Delphi удобна для тех, кому близок стиль ANSI.

**Тип Variant**  
    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.

**Вариантные значения**  
    При рассмотрении типа Record мы ознакомились 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 "проворен" и полезен в управлении данными разных типов. Переменным типа Variant можно присваивать любые значения любых целых, действительных, строковых и булевых типов. Для совместимости с другими языками программирования предусмотрена также возможность присвоения этим переменным значений даты/времени и объектов OLE Automation. 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. Можно произвольно задавать для выражении тип Variant в форме Variant (X).  
    В Object Pascal определены два особых значения Variant. Значение Unassigned применяется для указания, что вариантной переменной пока не присвоено значение какого бы то ни было 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 этого массива должны быть одного типа. На первый взгляд, это вполне естественное условие. Однако элементам массива можно присвоить и тип Variant! Тогда каждый элемент сможет содержать информацию разных типов, в том числе массив Variant. Как правило, вариантные массивы создаются с помощью процедуры VarArrayCreate.  
    Для передачи двоичной информации между контроллерами автоматизации OLE и серверами обычно применяются вариантные массивы с элементами varByte. Вариантные массивы типа varByte не могут подвергаться никаким преобразованиям. Нельзя также переформатировать содержащуюся 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.

**Процедуры обработки вариантных массивов**  
    В табл. 1.9 перечислены стандартные процедуры и функции обработки вариантных массивов, определенные в модуле System.

**Таблица 1.9.** Процедуры и функции обработки вариантных массивов

|  |  |
| --- | --- |
| **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 |

    В табл. 1.10 перечислены типы значении, которые можно присваивать вариантным переменным, и вариантные типы результата.

**Таблица 1.10.** Вариантные типы

|  |  |
| --- | --- |
| **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). 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.

**OLE Automation**  
    Вариантные переменные удобно применять для изменения свойств объектов OLE Automation и вызова методов этого объекта. Чтобы инициировать эту возможность, необходимо подключить модуль OleAuto.  
Синтаксис вызова метода или обращения к свойству объекта OLE Automation такой же, как вызова из созданного класса. Есть, однако, несколько важных отличии. Во-первых, вызов метода объекта OLE Automation происходит по схеме позднего связывания, т.е. компилятор не проверяет, существует ли данный метод и правильно ли определены типы параметров. Для компилятора приемлемы любой идентификатор метода и любое число параметров разных типов. А это означает, что при выполнении вызванного таким образом метода может произойти ошибка.  
    Что же касается идентификаторов методов объекта OLE Automation, то они могут содержать любые алфавитные символы из международного набора, в том числе а, ь и ш.