<http://tipidannih.narod.ru/ssilochnij.htm>

<https://studfile.net/preview/971901/> -- Для Turbo Pascal

# Ссылочный (указательный) тип данных

**Переменные - указатели**

**Графика в Delphi**

## Переменные-указатели

**Переменные-указатели** это особый тип переменных, которые не содержат значения, а указывают на них - на ту ячейку памяти, где они фактически располагаются. И хотя справедливо считается, что использование указателей может приводить к трудно контролируемым ошибкам в программе, всё же переменные - указатели это очень эффективный инструмент для управления объектами в оперативной памяти компьютера.

### Описание переменных – указателей

   Указатель описывается ключевым словом**Pointer**. По первой букве ключевого слова принято называть переменные - указатели с первой буквы **P**:  
  
  **var** PInteger: Pointer; *//не типизированный указатель*  
  
   Ключевым словом **Pointer** задаётся так называемый не типизированный указатель, по аналогии с не типизированным файлом. Не типизированный указатель содержит просто **адреc** некой ячейки памяти. Объект, располагающийся начиная с этой ячейки, соответственно может быть совершенно любого типа и размера.  
  
   Также в Delphi существуют и типизированные указатели. Они могут указывать на объект соответствующего типа и размера. Именно "могут указывать", потому что это, по прежнему, адрес одной - первой ячейки области памяти, где располагается объект. И далее его использование в программе зависит от программиста!

Итак, типизированный указатель описывается ключевым словом означающим данный тип, перед которым ставится значок **^**:  
  
var PInteger: ^Integer; //указатель на переменную целого типа  
PText: ^String; //указатель на переменную типа String  
  
   Также можно описать любой свой тип, и задать переменную-указатель данного типа:  
  
**type**

TMyType = Record  
   X: Integer;  
   S: **String**;  
 **end**;  
  
**var** PMyPointer: ^TMyType;  
  
   Естественно, можно определить тип, для описания через него переменных - указателей. Делается это в том числе и потому, что, например, в процедурах и функциях в качестве параметров можно использовать только заранее описанный тип данных. Например, следующее описание задаёт функцию с параметром, являющимся указателем ранее описанного типа, результат которой также является указателем данного типа:  
**type**

TPMyPointer = ^TMyType;  
  
**function** MyFunc(Point: TMyPointer): TMyPointer;  
  
   Ну вот, надеюсь, теперь вы не боитесь разнообразных вариантов описания таких переменных как указатели.

### Использование переменных – указателей

   Использование указателей предполагает:

1. [**присвоение значения указателю**](http://www.delphi-manual.ru/pointers.php#1)**;**
2. [**изменение значения указателя**](http://www.delphi-manual.ru/pointers.php#2)**;**
3. [**создание области памяти нужного типа и присвоение его адреса указателю**](http://www.delphi-manual.ru/pointers.php#3)**;**
4. [**запись значения в область памяти, адресуемой указателем, и чтение из неё**](http://www.delphi-manual.ru/pointers.php#4)**;**
5. [**освобождение области памяти, адресуемой данным указателем**](http://www.delphi-manual.ru/pointers.php#5)...

   Описанный указатель без присвоенного значения указывает на совершенно неопределённую ячейку памяти. Попытка использовать такой указатель чревата крахом программы. Поэтому всем указателям нужно явно присваивать значения.  
  
   **1.** Указателю можно присвоить значение другого указателя. В результате оба указателя будут указывать на одну и ту же ячейку памяти. Также указателю можно присвоить пустое значение с помощью ключевого слова  **nil**:  
**var** P1, P2: Pointer;  
**begin**  
  P1:=P2; *//Присвоение указателю значения другого указателя*  
  P2:=**nil**; *//Присвоение указателю "пустого" значения*  
**end**;  
  
   Указатель со значением **nil** не адресует никакой ячейки памяти и единственное, что с ним можно сделать - это сравнить с другим указателем или со значением **nil**.  
  
   **2.** Значение типизированного указателя можно увеличить или уменьшить на размер области памяти, занимаемой объектом данного типа. Для этого служат операции инкремента и декремента:  
**type** P: ^Integer;  
**begin**

*//увеличение значения указателя на 4 байта (размер типа Integer)*  
  inc(P);

*//уменьшение значения указателя на 4 байта (размер типа Integer)*

dec(P);

**end**;  
  
   Попытка выполнить операции  **inc**  либо  **dec**  с не типизированным указателем вызовет ошибку на этапе компиляции, так как компилятору неизвестно, насколько именно изменять значение указателя.  
  
   **3.** Процедурой **New** можно создать область памяти соответствующего типа и присвоить её адрес указателю (инициировать указатель):  
  
**var**  PInt: ^Integer;  
**begin**

*//Указатель****PInt****получает значение адреса созданной области памяти типа Integer*  
  New(PInt);

**end**;  
  
   Поскольку с областью памяти, созданной с помощью процедуры  **New**, не связана ни одна переменная, но там содержится реальное используемое значение, то можно считать, что это значение связано с некой "безымянной переменной". Обращаться к ней по имени переменной невозможно, а можно оперировать только используя указатель.  
  
   Также задать указателю адрес объекта можно с помощью операции, называемой "взятие адреса", которая обозначается значком **@.** При этом создавать область памяти уже не нужно, так как она создана предварительным описанием данного объекта:  
  
**var** MyVar: TMyType;*//Описание переменной, при этом выделяется область памяти соответствующего размера*  
    P: ^TMyType; *//Задаётся указатель соответствующего типа*  
**begin**  
  P:=@MyVar; *//Указатель получает адрес области памяти, занимаемой переменной MyVar*  
**end**;  
  
  
   **4.** Если область памяти уже создана и её адрес присвоен указателю, то в ячейку памяти, адресуемую данным указателем, можно записать значение объекта, соответствующего типу указателя. Для этого служит операция, обозначаемая также значком  ^, стоящим после имени указателя, например: **P^**. Эта операция называется "разыменование указателя". Также с помощью этой операции со значением в данной ячейке памяти можно делать всё что нужно:  
  
**var**  MyVar: Integer;  
     P: ^Integer;  
**begin**  
  P:=@MyVar; *//Указатель получает адрес области памяти, занимаемой переменной MyVar*  
  P^:=2; *//В ячейку памяти по адресу переменной MyVar записывается значение****2***  
  Form1.Caption:=IntToStr(P^+3); *//В заголовке Формы появится число****5***  
**end**;  
  
  
   С обычными переменными всё просто, но возникает вопрос, как получить значение по адресу указателя, если тип переменной - запись с несколькими полями? Аналогично:  
  
**type**

TMyRec = **Record**  
    N: Integer;  
    S: String;  
 **end**;  
  
**var**  MyRec: TMyRec;  
     PRec: ^TMyRec;  
  
**begin**  
  PRec:=@MyRec;*//Указатель получает адрес области памяти, занимаемой переменной MyRec*  
  PRec^.S:='Строка данных';*//С помощью указателя производится изменение строкового поля записи*  
  PRec^.N:=256; *//С помощью указателя производится изменение числового поля записи*  
**end**;  
  
   А теперь уберите ^ возле PRec: PRec.S:='Строка данных'; Вы увидите, что никакой ошибки ни компилятор, ни выполнение программы не показали! Выходит, выражения PRec^.S и PRec.S аналогичны.  
  
   Далее, а как получить значение, если указатель это элемент массива, например:  
  
**var** PArray: Array[1..100] of ^Integer;  
    X: Integer;  
  
   Ни PArray^[10], ни PArray[10]^ не являются правильными выражениями. Ну конечно, нужно использовать скобки:  
  
X:=(PArray[10])^;  
  
   **5.** Память, выделенную процедурой **New**, всегда нужно явно освобождать. Освободить область памяти, адресуемую указателем, инициированным с помощью **New**, можно процедурой Dispose:  
  
**var** MyVar: TMyType;  
    P: ^TMyType;  
**begin**  
  P:=@MyVar;  
  Dispose(P); *//Освобождение области памяти, адресуемой указателем P*  
**end**;  
  
   При выполнении процедуры **Dispose** указатель снова приобретает неопределённое значение, не равное даже **nil**, и его использование может привести к неопределённым результатам, даже к краху программы.

Процедуры GetMem, FreeMem самостоятельно. В отчёте обязательно привести примеры их использования.

### Динамические массивы

<http://www.delphi-manual.ru/lesson5.php>

**Delphi** можно использовать ***динамические*** массивы, количество элементов в которых допускается изменять в зависимости от требований программы. Это позволяет экономить ресурсы компьютера, хотя работа с такими массивами происходит гораздо медленнее. Описываются динамические массивы аналогично статическим, но без указания диапазона индексов:  
  
   **type**

TDinArray = **array of** Integer;  
    **var** A : TDinArray;

После создания в динамическом массиве нет ни одного элемента. Необходимый размер задаётся в программе специальной процедурой  **SetLength**, например, массив из ста элементов:  
  
   **begin**  
    SetLength(A,100);  
   **end;**  
  
 Нижняя граница динамического массива всегда равна нулю. Поэтому индекс массива **A** может изменяться от **0** до **99**.  
**Многомерные** динамические массивы описываются именно как **массивы массивов**. Например, двумерный:

**type** T3DinArray = **array of array of** Integer;  
   **var** B : T3DinArray;  
  
В программе сначала задаётся размер по первому измерению (количество столбцов):

|  |
| --- |
| SetLength(B,3);  Затем задаётся размер второго измерения **для каждого** из трёх столбцов, например:    SetLength(B[0],3);    SetLength(B[1],2);    SetLength(B[2], 1);  Таким образом создаётся треугольная    матрица:  **B00  B10  B20**  **B01  B12**  **B02**  Чтобы освободить память, выделенную динамическому массиву, нужно массиву как целому присвоить значение **nil**:     B := **nil;** Ключевое слово **nil** в Delphi означает отсутствие значения.  **[Динамический массив в Delphi - Конспект](C:\\Users\\S\\AppData\\Local\\Packages\\microsoft.windowscommunicationsapps_8wekyb3d8bbwe\\LocalState\\Files\\S0\\4\\Attachments\\Динамический массив в Delphi - Конспектhttps:\\konspect.blogspot.com › dynamic-array-in-delphi)**  [https://konspect.blogspot.com › dynamic-array-in-delphi](C:\\Users\\S\\AppData\\Local\\Packages\\microsoft.windowscommunicationsapps_8wekyb3d8bbwe\\LocalState\\Files\\S0\\4\\Attachments\\Динамический массив в Delphi - Конспектhttps:\\konspect.blogspot.com › dynamic-array-in-delphi)  Как только динамический массив был распределен, вы можете передавать массив стандартным функциям **Length**, **High**, **Low** и **SizeOf.** Параметр этих функций – имя динамического массива.  Функция **Length** возвращает число элементов в динамическом массиве.  **High** возвращает самый высокий индекс массива (то есть **Length**- 1).  **Low**возвращает 0.  В случае с массивом нулевой длины наблюдается интересная ситуация:  **High** возвращает -1, а **Low** - 0, получается, что **High**меньше  **Low.**  Функция **SizeOf**всегда возвращает 4 - длина в байтах  памяти указателя на динамический массив.  Например, для динамического массива А его параметры:    iHigh   := High (A);    iLow    := Low  (A);     iLength := Length (A);     iSizeOf := SizeOf (A);  Доступ к данным динамических массивов с помощью низкоуровневых процедур типа **ReadFile**или **WriteFile** , или любых других подпрограмм, получающих доступ сразу ко всему массиву, часто выполняется неправильно. Для  статического массива - в противоположность динамическому массиву переменная массива тождественна его данным. Для динамического массива это не так - переменная  это указатель. Так что если вы хотите получить доступ к данным динамического массива - вы не должны использовать саму переменную массива, а использовать вместо неё первый элемент массива.  правильно WriteFile(FFile, A[0], Length(A), dwTemp, nil);  неправильно WriteFile(FHandle, A, Length(A), dwTemp, nil);  Рассмотрим  пример присваивания динамических массивов одного другому  **var**  **da\_A,da\_B: array of integer;**  **begin**  **SetLength(da\_A,2);**  **SetLength(da\_B,2);**  **da\_A[0]:=2;**  **da\_B[0]:=3;**  **da\_A:=da\_B;**  **da\_B[0]:=4;**  **end;**  После этих манипуляций da\_A[0] равно 4. Дело в том , что  при присвоении da\_A:=da\_B не происходит копирование т.к. da\_A, da\_B, это всего лишь указатели на область памяти. Для копирования необходимо использовать функцию **Copy.** |

Рассмотрим пример копирования динамических массивов с использованием функции **Copy**:

**var**

**da\_A, da\_B: array of integer;**

**begin**

**SetLength(da\_A,2);**

**SetLength(da\_B,2);**

**da\_A[0]:=2;**

**da\_B[0]:=3;**

**da\_A:=Copy (da\_B);**

**da\_B[0]:=4;**

**end;**

После этих манипуляций da\_A[0] равно 3. После функции **Copy** da\_A и da\_B указывают на разные области памяти, поэтому при изменении da\_B в da\_A ничего не происходит и его значения остаются неизменными**.**

### Функция Copy

http://www.delphibasics.ru/copy.php

[Функция описана в модуле](http://helpapi.narod.ru/OsnovDelphi/072.htm) **[System](http://helpapi.narod.ru/OsnovDelphi/072.htm)**[. Функция](http://helpapi.narod.ru/OsnovDelphi/072.htm)**[Copy](http://helpapi.narod.ru/OsnovDelphi/072.htm)**[имеет 2 формы.](http://helpapi.narod.ru/OsnovDelphi/072.htm)

[В первом случае она создает новую строку из части существующей строки. Во втором - она создает новый массив из части существующего массива.](http://helpapi.narod.ru/OsnovDelphi/072.htm)Пример для копирования строки:

**var**  
  Source, Target : string;  
**begin**  
  Source := '12345678';  
  Target := **Copy**(Source, 3, 4);  
  ShowMessage('Target : '+Target);  
**end;**

Первый символ строки имеет индекс = 1.  
Возвращает строку состоящую из указанных в **Count** количестве символов, начиная с **StartChar** символа.  
Если до конца строки меньше символов, чем указано в **Count**, то копируются все символы, начиная с **StartChar** и до конца строки. [Target : 3456](http://grechkinapv.appmat.ru/Semestr2/PDF/Part-2-Lection-4-DynArray.pdf)

**Пример кода - копирование массива**:

**var**  
  i : Integer;  
  Source, Target : array of Integer;  
  
**begin**  
  SetLength(Source, 8);  
 **// Формирование динамического исходного массива**  
  for i := 1 to 8 do

**// Запомните, что массивы начинаются с индекса = 0**  
    Source[i-1]:=i;

//**Source =1 2 3 4 5 6 7 8**  Target := **Copy**(Source, 3, 4);

**// Показываем созданный массив**  
for i := 0 to Length(Target) -1 do

ShowMessage('Target['+IntToStr(i)+']:

'+IntToStr(Target[i]));

**end;**

Target[0] : 4  
Target[1] : 5  
Target[2] : 6  
Target[3] : 7

Первый элемент массива имеет индекс = 0.  
Возвращает массив состоящий из указанных в **Count** количестве элементов, начиная с индекса **StartIndex** .  
Если до конца массива меньше элементов, чем указано в **Count**, то копируются все элементы начиная с **StartIndex** и до конца массива.

Удалить из памяти динамический массив можно одним из следующих способов:

* A := nil;
* Finalize(A);
* SetLength(A,0).

### Проблема потерянных ссылок

1. При работе с динамическими переменными следует освобождать выделенные области памяти сразу же, как только необходимость в них отпадает, иначе «**засорение » памяти такими переменными приводит к быстрому её исчерпанию.**
2. Вторая проблема может возникнуть при создании локальных динамических переменных. При завершении подпрограммы локальные переменные уничтожаются (очищается стек, где они находились). Но если они не уничтожаются с помощью Dispose или FreeMem, то память, выделенная в куче, становится не доступной, что может привести **к быстрому её исчерпанию.** Вот пример такой процедуры:

type

PPerson =^Person;

Person = record

. . .

end;

procedure GetPerson;

var

P : PPerson;

begin

New(P);

end;

Процедура New(P) в GetPerson выделит память для динамической переменной P типа PPerson.

При выходе из GetPerson локальная переменная Р перестанет существовать. Однако, область памяти, отведённая в процессе работы GetPerson, продолжает существовать, т.к. освободить её можно только явно, используя процедуру Dispose(Р). Таким образом, после выхода из GetPerson отсутствует доступ к динамической переменной P, а следовательно и к памяти, отведённой под эту переменную.

Нужно запомнить, что при выходе из блока необходимо или освободить все созданные в нём динамические переменные или присвоить ссылки на них глобальным переменным.