Pointers in Delphi

An introduction to pointer data type in Delphi. What are pointers, why, when and how to use them.

One of the things beginners in Delphi (and programming in general) find most difficult to understand is the concept of pointers. The purpose of this article is to provide an introduction to pointers and their use to Delphi beginners.

Even though pointers are not so important in Delphi as the are in C or C++, pointers are such a "basic" tool that almost anything having to do with programming must deal with pointers in some fashion. For that reason, you will read that "a string is really just a pointer" or that "an object is really just a pointer" or "event handler such as OnClick is actually a pointer to a procedure".

Pointers can be a very powerful aspect of a language. Unfortunately, pointers are also frequently behind much of the complexity of the languages that use them. The Object Pascal language of Delphi took a big step toward removing the complexity of pointers by hiding them in many cases.

Pointer as data type

So what is a pointer? Simply put, a pointer is a variable that holds the address of anything in memory.
To concrete this definition, just keep in mind the following: everything used in an application is stored somewhere in the computer's memory, a pointer to that item simply points the program to that memory location. Because a pointer holds the address of another variable, it is said to point to that variable.

We can consider a pointer as a means to hold a data address at runtime. It can point to different variables in turn, even to unnamed variables which are alive only for a particular run-time period.

As with all Delphi types, we must declare a pointer before we can use it. Most of the time pointers in Object Pascal point to a specific type:

```
var
 iValue, j : integer;
 pIntValue : ^integer;
begin
  iValue := 2001;
  pIntValue := @iValue;
  ...
  j:= pIntValue^;
end;
```

The syntax to declare a pointer data type uses a caret (^). In the code above iValue is an integer type variable and pIntValue is an integer type pointer. Since a pointer is nothing more than an address in memory, we must assign to it the location (address) of value stored in iValue integer variable. The @ operator returns the address of a variable (or a function or procedure as will be seen later in this article). Equivalent to the @ operator is the Addr function. Note that pIntValue's value is not 2001.

In the code above the pIntValue is a typed integer pointer. Good programming style is to use typed pointers as much as you can. The *Pointer* data type is a generic pointer type - represents a pointer to any data.

Note that when "^" appears after a pointer variable it dereferences the pointer; that is, it returns the value stored at the memory address held by the pointer. In the code above (after it) variable j has the same value as iValue. It might look like this has no purpose when we can simply assign iValue to j, but this peace of code lies behind most calls to Win API.

NILing pointers
Unassigned pointers are dangerous. Since pointers let us work directly with computer's memory, if we try to (by mistake) write to a protected location in memory we could get a GPF error. This is the reason why we should always initialize a pointer to a special value of NIL. The reserved word nil is a special constant that can be assigned to any pointer. When nil is assigned to a pointer, the pointer doesn't reference anything. Delphi presents, for example, an empty dynamic array or a long string as a nil pointer.

Character pointers
The fundamental types PAnsiChar and PWideChar represent pointers to AnsiChar and WideChar values. The generic PChar represents a pointer to a Char variable. These character pointers are used to manipulate null-terminated strings. Think of a PChar as being a pointer to a null-terminated string or to the array that represents one. For more on null-terminated strings go see: "String types in Delphi". Have in mind that long-string variables are implicitly pointers.

Pointers to records
When we define a record or other data type, it's a common practice also to define a pointer to that type. This makes it easy to manipulate instances of the type without copying large blocks of memory. In fact, any data type that requires large, dynamically allocated blocks of memory uses pointers.

The ability to have pointers to records (and arrays) makes it much more easier to set up complicated data structures as linked lists and trees. What follows is an example of the type declaration for a simple linked list:

```
type
  pNextItem = ^LinkedListItem
  LinkedListItem = record
    sName    : String;
    iValue   : Integer;
    NextItem : pNextItem;
end;
```

The idea behind linked lists is to give us the possibility to store the address to the next linked item in a list inside a NextItem record field. For more on data structures consider the book: "Ready to run Delphi Algorithms".

Procedural and method pointers
Another important pointer concept in Delphi are procedure and method pointers.

Pointers that point to the address of a procedure or function are called procedural pointers. Method pointers are similar to procedure pointers. However, instead of pointing to stand-alone procedures, they must point to class methods. Method pointer is a pointer that contains information about the name of the method that is being invoked as well as the object that is being invoked. To see some function pointer in action go see: "Dynamic World of Packages". We'll talk more about such pointers in some of the future articles.

Pointers and Windows API

The most common use for pointers, in Object Pascal, is interfacing to C and C++ code, which includes accessing the Windows API. Windows API functions use a number of data types that may be unfamiliar to the Delphi programmer. Most of the parameters in calling API functions are pointers to some data type. As stated above, we use null-terminated strings in Delphi when calling Windows API functions. In many cases when an API call returns a value in a buffer or a pointer to a data structure, these buffers and data structures must be allocated by the application before the API call is made.
For example, take a look at the SHBrowseForFolder Windows API function. This function is used to invoke a Windows system dialog used to browse for files and folders on users hard drive as well as network computers and printers.

Pointer and memory allocation

The real power of pointers comes from the ability to set aside memory while the program is executing. I would not like to bother you with heaps and memory programming, for now the next peace of code should be enough to prove that working with pointers is not so hard as it may seem.

The following code is used to change the text (caption) of the control whose Handle is provided.

```
procedure GetTextFromHandle(hWND: THandle);
var pText : PChar; //a pointer to char (see above)
    TextLen : integer;
begin
{get the length of the text}
TextLen:=GetWindowTextLength(hWND);

{alocate memory}
GetMem(pText,TextLen); // takes a pointer

{get the control's text}
GetWindowText(hWND, pText, TextLen + 1);

{display the text}
ShowMessage(String(pText))

{free the memory}
FreeMem(pText);
end;
```