Lesson 15: Article 3: Pointers

Pointers are another way of storage. So far, in the lessons that you have might read, we used only variables to store data items.

Variables occupy memory locations to store data items. On the other hand pointers only occupy memory space for themselves, but do not store data items. They store the location of the item that they are pointing to. They indirectly access the data that it is set to point to. Before a pointer can be used, it should be set to point to a memory location otherwise we get into trouble. So always remember to initialise a pointer and set it to point to a memory location. But, how can we set a pointer to point to a memory location? This is to be shown in the next section.

Using Pointers

In the implementation below, I will show you how to intialise a pointer and use it, after which I explain what is going on.

```
Program Pointers_Example;
Var
         int_ptr : ^Integer;


Begin
         new(int_ptr);
         int_ptr^ := 50;
         Writeln('The pointer is pointing to a memory location storing', int_ptr^);
         int_ptr^ := int_ptr^ + 1;
         Writeln('Now it is increased by 1:', int_ptr^);
         dispose(int_ptr^);
         Writeln('The pointer has no more significance: ', int_ptr^);
End;
```

In the line,

Var int_ptr : ^integer;

we are decalring an integer pointer. It so called because we are using integer as data type. A pointer may be any of the existing data types, even user defined data types. We use the (^) character to indicate a pointer declaration.

The line:

new(int_ptr);

Dynamically creates and allocates space for *int_ptr*. Before this line, the integer pointer was simply resident in memory without pointing to any memory location. Now, after we have dynamically created the pointer, it is set to point to a memory location chosen by the OS.

The line:

int_ptr^ := 50;

stores the number 50 in the memory location pointed to by *int_ptr*.

Finally the integer pointer is disposed of when calling dispose(), passing the pointer as argument to the function. When a pointer is disposed of, it won't be possible to use it again (i.e. its link to the memory location that it has been previously pointing to is destroyed) unless re-created dynamically using new().

A pointer may be assigned a ***nil*** value when it is not being used for the moment but is planned to be used later. One can also test if a pointer is *nil* using the statement

if my_ptr = nil then ...

This will return true if *my_ptr* is set to point to nothing.

Pointers may in effect be set to point to the same location. This is done by using the assignment statement just as when we do an assignment of one variable to the other. When assigning two pointers of the same type, the assignment is bi-directional. Here's how we do an assignment of two similar typed pointer variables:

int_ptr1 := int_ptr2;

Assuming that both variables are integer pointers, they will now point to the same location. From now on, one either of the two integer pointers is assigned a value, they will change the memory location accordingly.

Consider the following (keeping in mind that both integer pointers are assigned to each other):

```
int_ptr1^ := 10;
writeln(int_ptr2^);
```

The output of the writeln command is 10 since the memory location that is pointed to by int_ptr2 (which is also pointed to by int_ptr1) is changed by int_ptr1.

If we had to dispose int_ptr1^ before we actually assign it to int_ptr2, int_ptr1^ will lose the link to the memory location it has been previously pointing to. When it will be assigned to int_ptr2, the link is re-created and is set to point to the same location pointed to by int_ptr2.

Eventually, more than two pointers can point to the same memory location, however, despite the fact that more than one pointer can point to the same location, one should keep in mind the careful organization and memory storage of the values stored in a memory location by the pointers.

A pointer is stored in memory just like the way other variables are stored. Consider the following:

Memory Location	**Type**	**Value**
06A8h	Variable (Integer)	5
06B0h	Variable (String)	Hey there!
06B2h	Pointer (Integer)	4FE4h

The memory location is where the variable/pointer is situated in memory. The operating system keeps track of where all these variables are located for later reference. As you can see, each memory location stores data in it. The pointer, obviously stores the memory location it is pointing to and not the value. It also indicates that it is pointing to an integer data type. When you obtain a value via a pointer, the operating system looks at the pointed location and acquires the value from there - this is called ***dereferencing.***

A pointer stores a memory address (in hexadecimal) which is the location pointed by the latter. Pointers are very prone to illegal access of memory locations in memory and can result to what is commonly referred to as ***Segmentation Error*** (in Linux). When this happens in Windows, an illegal exception occurs and program is terminated. This is called memory protection, in which case the operating system won't allow a program to access memory outside its pre-allocated memorary range. This was not so in early versions of Windows operating system and accessing illegal memory locations would result in corrupting the windows that is loaded into memory and the whole system stalls. There is a lot of more advanced uses of pointers. This will be demonstrated in the Linked Lists article.