

# Re: pointer syntax

---

*Source:* <http://coding.derkeiler.com/Archive/Delphi/comp.lang.pascal.delphi.misc/2005-11/msg00219.html>

---

- *From:* "Maarten Wiltink" <[maarten@xxxxxxxxxxxxxxxxxxxxx](mailto:maarten@xxxxxxxxxxxxxxxxxxxxx)>
  - *Date:* Wed, 23 Nov 2005 10:58:31 +0100
- 

"swansnow" <[schultz@xxxxxxxxxxxxxxxxxxxxx](mailto:schultz@xxxxxxxxxxxxxxxxxxxxx)> wrote in message  
[news:1132687818.534993.261370@xxxxxxxxxxxxxxxxxxxxx](mailto:news:1132687818.534993.261370@xxxxxxxxxxxxxxxxxxxxx)  
> Maarten said:

- >> If so, just pass the object by value, and call the method on it.
- >
- > When you say "Pass by value" does that mean to use a "var" parameter?

Other way around. It's in the help.

[...]

- > I learned C way back when, then a bit of C++, but my most recent
- > "other" language is Java. In Java, you have objects or primitives
- > (int, char, and so forth are primitive). If you pass an object, you're
- > really passing a reference (which is like an old-school pointer, you
- > just can't dereference it). If you modify the object, then you'll see
- > the changes persist. If you pass a primitive, you're actually creating
- > a copy of the primitive's value and passing that.

Or you can pass a primitive by reference ("ref" modifier in Java?) and changes will persist outside the function, too.

Parameter passing works the same in virtually all languages. You can either pass by value, or by reference. Syntax differs but the intent and the mechanisms do not.

- > My understanding of Delphi follows. Please correct me where I'm wrong
- > :)
- >
- > \* Delphi makes a distinction in syntax between using objects and using
- > non-objects like records

I'd sooner say it makes `_no_` syntactical distinction. But objects are implicitly references. Define a classtype variable and a four-byte pointer is allocated. Then you have to instantiate an object and store the reference to it in that variable. From then on, dereferencing the variable is done implicitly whenever required.

## Re: pointer syntax

This is the same as in Java and that's not coincidental. Many programming languages differ only by expressing the same things in different words.

- > \* "normal" passing of non-objects is by value, and if you change them
- > inside a procedure, the changes go away when the procedure exits.

Yes. Because you're changing a copy of the value, not the original value.

- > \* If you say "var" in the argument list, then passing the non-object
- > allows the procedure to make changes that persist. Kind of like having
- > multiple return values.

Kind of. Result is a hidden `_out_` parameter. Same mechanism as `var`, different intent. An out parameter starts uninitialised.

- > \* If you have a non-object data structure, like a record, you can
- > explicitly use pointers to avoid passing by value, and to allow
- > modification of the structure inside a procedure (like old-school C
- > programming)

Why would you want to avoid passing by value? (Because if it's a big record, it takes up a lot of stack space.) Avoid premature optimisation instead. (Use a `const` parameter if applicable.)

C does not offer reference parameters in the language, so you have to build them yourself with pointers. Just don't do that in Pascal. If you mean a `var` parameter, write a `var` parameter.

- > \* If you have a non-object data structure, with "var" in the parameter
- > list, you're passing by value, which means you're actually creating a
- > copy of the data structure and passing that it. If you make changes to
- > it, the changes get passed back, and so they persist. You can avoid
- > using pointers (and their messy syntax) if you do things this way.

That `_would_` be messy, if it were what happened. It isn't. Please forget about this whole horrible scheme.

- > \* If you have an object (an instance of a class), then you get stuff
- > handled for you automatically, like in Java. You are passing references
- > around, so changes persist. If you don't want changes to persist, then
- > you have to explicitly create a copy (using a `create` statement, with
- > perhaps a copy constructor) and work on that.

Yes. But as you undoubtedly know, once objects have interactions with the outside world, copying objects is a can of worms. In my experience,

Re: pointer syntax

copying objects is relatively rare.

> It seems to me that I'm missing something...

Yes. You have the "var" modifier on parameters exactly backwards. The default is call by value. Var makes a parameter call by reference.

> Question:

> If I have a TTable object, t, and I say:

>

> var

> t2 :TTable;

> begin

> t2 := t;

>

> ...what is the result? Do I now have two references pointing to the

> same object? If t's state is dsEdit, will t2's state be dsEdit, too?

Yes. You've created an alias. The two states will be the state of the same object.

Groetjes,  
Maarten Wiltink

---

• **Follow-Ups:**

- ◆ **Re: pointer syntax**  
◇ From: swansnow

• **References:**

- ◆ **pointer syntax**  
◇ From: swansnow
- ◆ **Re: pointer syntax**  
◇ From: Maarten Wiltink
- ◆ **Re: pointer syntax**  
◇ From: swansnow

- Prev by Date: **Re: pointer syntax**
- Next by Date: **copy one character from a string.**
- Previous by thread: **Re: pointer syntax**
- Next by thread: **Re: pointer syntax**
- Index(es):
  - ◆ **Date**
  - ◆ **Thread**