|  |
| --- |
| Circle Language Spec |

## Pointers

### Brainstorm

The 'pointer-to-pointer' issues seem to be a bit spread over the chapters. When a chapter is explained, afterwards it seems to evaluate how things would look in pointer-to-pointer situations. The idea is that all of those pointer-to-pointer situations might be put here in this chapter instead. Topics like objects, classes, interfaces, assignment, seem to able to live without thinking about pointer situations, and pointers just seems a single problem area that might be desirable to cover separately.

It might be worth highlighting there may be different interpretations of pointers, lines and their direction. They seem to be non-competing. Here is an attempt to summarize some of them:

* Interpretation 1: Line direction would not matter, only aspect correspondence would matter.
* Interpretation 2: Direction tends to point outwards, if inward, this would be denoted with an access symbol.
  + The notational choice would be arbitrary and carry no special meaning.
* Interpretation 3: Directions tend to point outwards, if inward, this would be denoted with an access symbol, like previously.
  + Inward directions would actually be more 'active' redirections/accesses: Pointer-to-pointer redirections, getter accesses, calls to procedures returning an object, etc.
  + Outward directions, would be more passive. They might represent 'simple' pointers, not represent getter calls or anything, more like indications of aspect correspondance.
* Interpretation 4: All symbols would be pointers, kind of like in some languages objects might be accessed through singly-redirected object references (C# assumably).
* Interpretation 5: There would be one symbol in the diagram, that represents the actual object, not a pointer to it.
  + It might be found by first following all outward redirections, then all the inward ones.
  + Where it ends, might be the 'target' symbol: The actual target of the redirections that might be said to be represent the actual object, rather than just a reference to it.

### Brainstorm Ref-Ness

Another topic that might be covered, is a comparison with other languages (even though one of the strategic items is to not try and compare so much in this text, with the idea that 'where would it end?') An exception to the rule could be made here to add a comparison to other language's ref-ness, because Circle seems to be 'make a mockery' of the concept ref-ness in a way. C# or C++ seem to be specific about ref-ness. (C++ might make you specify asterisks \*\* to indicate how many redirections a pointer variable makes; C# and .NET seem to assign intrisic importance to defining parameters as ref or out and what other 'ref-nesses' have you? Anyway, they seem quite specific.) Circle however, seems to make a 'mockery' out of this, because all you need to do is add a line and the ref-ness changes. And the ref-ness does not seem to be specified near the start of the pointer redirection, but you might arbitrarily let redirections be added by the thing you are pointing to. 'mockery' is a meant a bit humoristically here, of course. It is just a notation. If the diagrams might represent something from C#, rules are probably just bound by what you can do in C#. You simply might not be able to add more redirections, or might not validly specify something with not enough redirection. Getter accesses in C# might actually be C#'s own embodiment of indeterminate ref-ness. Or depending how lightly you might want to apply the diagram language, it might not really matter that much, this ref-ness issue and these diagrams. But what might become a splinter in your brain, is that Circle does not seem to have a notation (yet) to specify fixed ref-ness. And what might rub some against the fur, is that Circle seems to like indeterminate ref-ness while some might hold determinate ref-ness in great value perhaps. The notion that there are these ideas about that, might justify thinking about it and perhaps describing a way to elegantly solve it or perhaps find a way to live with things the way they are.