**Loop/Conditional Constructs in Shackleton**

This document outlines some of the ideas for implementing support for loop and conditional constructs in Shackleton. One major goal was to attempt to change the Osaka structure implementation as little as possible. There will be some changes required in order to implement these more complex constructs, but the aim is to make those changes as non-intrusive to the current implementation as possible.

My inspiration and intuition came from the way that DNA bunches up on and folds against itself naturally when representing certain proteins and other structures. What if we can that concept and represent these constructs in a similar way? The whole point is to allow these constructs to be represented in an intuitive way to a normal user, while still be unmistakably linear in its construction.

We could represent an if…then statement as something like this:

Which, once unwrapped would be represented like so:

In a similar vein, an if…then…else statement would look something like this:

And when unwrapped would be represented like so:

In the above visualization, nodes with a light green color represent regular nodes that will be executed as they are and are not explicitly linked to other nodes in any way outside of the overall doubly linked list structure. The nodes with the darker green color represent the nodes that are included in the structural elements of the if…then (…else) construct. These nodes would need some form of link to the other nodes they share the construct with.

For this to be implemented in the current Shackleton framework, each node would have an optional field that indicates a choice to be “linked” to another node. This linkage indicates that they are a part of the same if…then (…else) construct. It would almost be a phantom doubly linked list, one that only links things in the same construct together and does not include any other nodes. During evolution, linked nodes cannot be separated from one another and are not good candidates for mutation. However, segments of code within two nodes of any construct can freely be mutated and crossed over with any other segment. I think this had promise for limiting some of the “bad” changes that may arise from the evolutionary process as well.

Implementing the loop conditions in this way would require a relatively non-intrusive way to include these in the Osaka structure and allow for the structures to still be treated as a fully linear representation.