/*
 * Authors: Olaolu Emmanuel && Jordan Lyon
 */

<u>SLCreate</u>
space: size of two function pointers and a struct pointer. O(1)
efficiency: initialize all variables in struct. O(1)

<u>SLDestroy</u>
space: size of one two struct pointers. O(1)
efficiency: frees each node and then the struct. O(n) where n is the number of nodes.

<u>SLInsert</u>
space: size of one node. O(1)
efficiency: worst case is end of list insertion which is O(n)

<u>SLRemove</u>
space: size of two struct pointers. O(1)
efficiency: worst case is not finding the target which is O(n)

<u>SLCreateIterator</u>
space: size of one Iterator struct. O(1)
efficiency: O(1)

<u>SLDestroyIterator</u>
space: only space for a few pointers. O(1)
efficiency: iterator is of constant size O(1)

<u>SLNextItem</u>
space: only requires pointer to the data item O(1)
efficiency: uses the previous item's next-node value to find the next node. O(1)

<u>SLGetItem</u>
space: returns pointer. O(1)
efficiency: pointer is already in the struct O(1)

<u>Implementation Notes</u>
- If the user attempts to delete a node which an iterator is currently “on” (“on” meaning that that node was the most recent node returned from that iterator by SLNextItem) then we do not perform that deletion and instead return 0. We do so because if the user were to delete the node an iterator is on and then delete the node after that, then SLNextItem would return garbage since the next node has been deleted.