A stack is useful for this problem because it can be used to look at the string character by character. Whether it be pushing ‘A’s and popping ‘B’s to make sure they are of equal number, or using the stack to look for repletion in a string. Storing a sample in a stack and peeking/popping to compare the rest of the string for repetition.  
  
I chose to implement the stack using a linked list so the size of the stack could easily be extended. I used a pointer to keep track of the top of the stack. Looking at the top\*->value when I needed to peek or pop from the top. Nodes had a pointer pointing to the previous node vs the normal linked list where it points to the next node. This way I could ‘pile’ nodes on top of another like a normal stack.

**Time Complexity**  
Constructor: O(1)  
isEmpty(): O(1)  
pop() :O(1)  
push(T item): O(1)  
peek(): O(1)  
L1(char\* inputString): O(N)  
L2(char\* inputString): O(N)  
L3(char\* inputString): O(N)  
L4(char\* inputString): O(N)