**Assignment Set Number: Problem Set 04**

**Group Name: GROUP 46**

**Contribution Table:**

| **Group No** | **Name** | **Student Email ID** | **% Contribution** |
| --- | --- | --- | --- |
| **46** | **NAVANEETTHA S** | **2021SC04149@wilp.bits-pilani.ac.in** | **Equal(100%)** |
| **46** | **SHALINI G NAIR** | **2021SC04148@wilp.bits-pilani.ac.in** | **Equal(100%)** |
| **46** | **YAMINI K** | **2021SC04150@wilp.bits-pilani.ac.in** | **Equal(100%)** |

Design Document Problem Set 04

## Function checkIfPalindromUsingStack:

**Data Structure Used**: STACK

The stack class is Implemented using Linked Lists in our python file

**Input**: str which is of type String

**Output**: If the string is Palindrome it returns the string itself, else Not a Palindrome

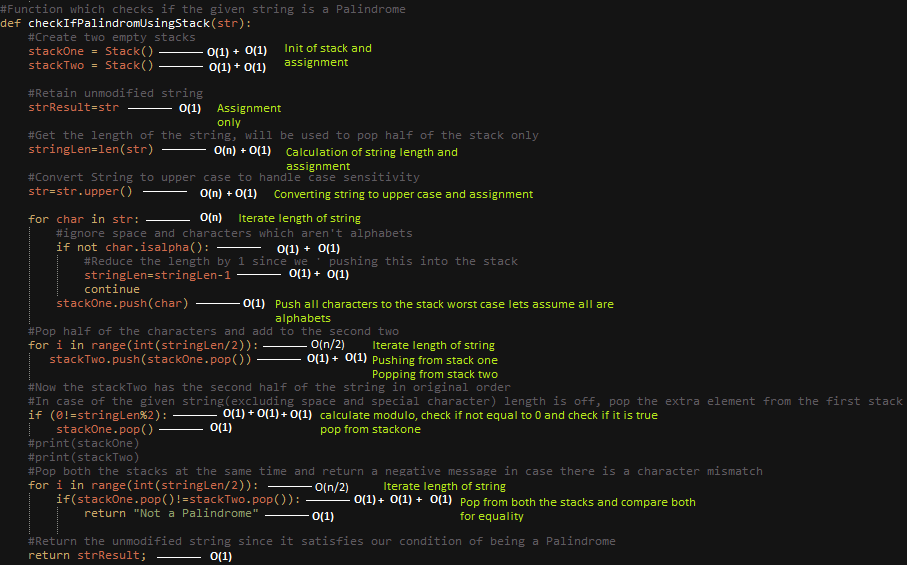
**Note: We have assumed that special characters and spaces should be ignored and only alphabets are to be considered to check for the palindrome property**

## Time Complexity: O(n) where n is the length of the string

The stack data structure has linear time complexity for creation of the stack also both insert(push) and delete(pop). From the figure attached we could infer the below:

| Operation | Complexity |
| --- | --- |
| Initialization of Stacks | O(4) |
| Pre-processing of String | O(2n+4) |
| Pushing to Stack One | O(5n) |
| Popping half the elements | O((n/2)\*2)=O(n) |
| Popping extra element | O(4) |
| Popping and comparing | O((n/2)\*4)=O(2n) |
| Return statement | O(1) |
| Total | O(10n+13) |

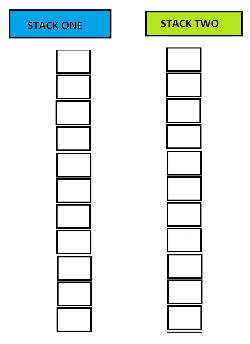
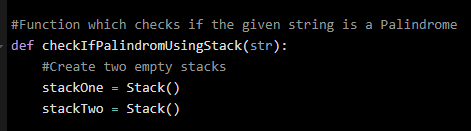
O(10n+3) 🡪 Thus we conclude the algorithm uses O(n) complexity



## Steps of this function which checks if the String is a Palindrome or Not is as below:

Explained using the example: **Borrow or rob?**

1. Initialize two Empty Stacks

1. Get the length of the string and store in a variable

stringLen=**14**

1. Retain the original string in another string

strResult= **Borrow or rob?**

* This is done to have the original string to be returned by the function in case if it is a palindrome, the string will be processed in upcoming steps

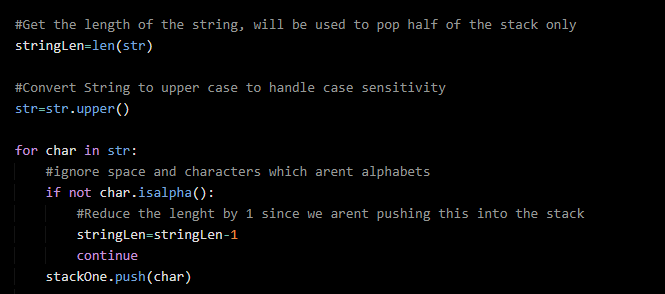
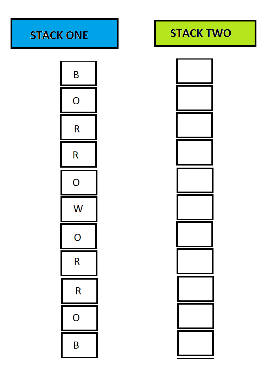
1. Convert the string to Upper case

str= **BORROW OR ROB?**

* This step is to handle case sensitive comparison

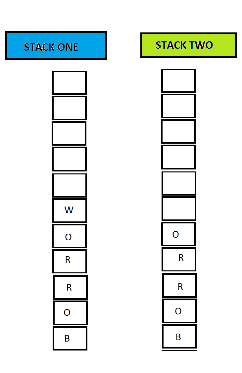
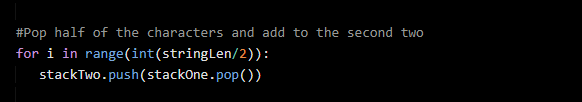
1. Push the characters which are alphabets one by one into stackone, if they aren’t alphabets. In our case two spaces and a “?” will be ignored. So at the end of this step:

stringLen=**11**



1. Pop Half of the characters and push them into second stack

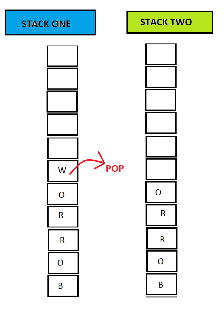
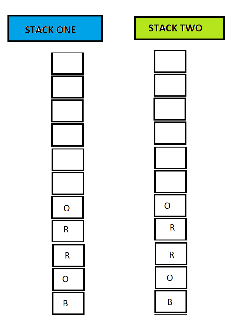
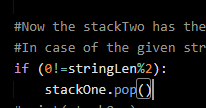
stringLen/2 = **5**

1. Since there are odd number of characters which are alphabets we need to pop the one extra element from the stack

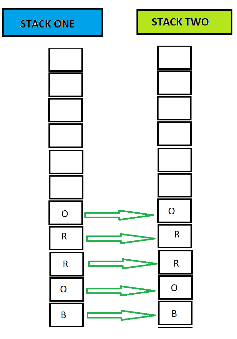
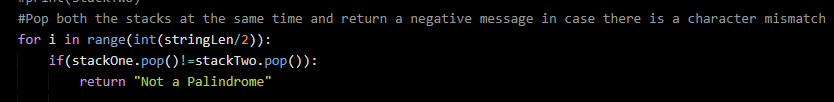
stringLen%2 = **1**

0!=stringLen%2 🡪 **True**

1. For half of the string length times pop both the stacks and compare the popped elements

stringLen/2 = **5**

* In this step in case there is a mismatch “Not a Palindrome” will be returned

1. At the end of the for loop return the retained string

strResult= **Borrow or rob?**

# Justification for the choice of Data Structure: Stack follows Last in First Out(LIFO) approach by leveraging this behaviour we could push the characters of a given string and when popped it would be reversed. When the popped characters (only half of the characters) from the first stack (with reversed order) is pushed into another stack it will be persisted in its original order. At the end of these two steps we would have two stacks one with original order of second half and other with reversed order of first half. We could pop the two stacks at the same time and compare if the characters are the same. This is the property of a palindrome where the original and the reversed string are the same. Hence Stack is a good option for checking this property of a Palindrome. Other approaches:

There could be multiple ways to solve a problem, each of which has its own advantages and disadvantages. The choice of which algorithm to use is made based on some parameters, two such important parameters are space and time complexity. Below are some of the approaches which can be used to check if a string is a palindrome or not:

1. **Doubly Ended Queue:** We could enqueue all the characters of a String to a double ended queue. Dequeue the characters from both ends and compare for equality. If there is a mismatch we could conclude the String is not a palindrome. This operation can be done until the Queue is empty.

**2. Two pointer approach:**We could have two pointers, one at the beginning of the String (say start) and other at the end of the String (say end). Compare the characters at both the pointers, in case there is a mismatch we can conclude the String is not a Palindrome. We should keep incrementing the start pointer and decrementing the end pointer until the pointers cross each other.

**3. Recursion:**We could solve this by using a recursive approach, where we can compare the character at the start of the string with the end of the string. And the function could be called recursively with the input string where we trim the start and the end character from it. In case we find a mismatch we could conclude that the string is not a palindrome.