A Palindrome Checker Program

This program determines if a given string is a palindrome. A palindrome is something that reads the same forwards and backwards. For example, the words “level” and “radar” are palindromes. The program imports the stack module developed in the previous section. This program utilizes the following programming feature:

- ➤ Programmer-defined module

Example execution of the program is given here:

Program execution ...

```
This program can determine if a given string is a palindrome

(Enter return to exit)
Enter string to check: look
look is NOT a palindrome

Enter string to check: radar
radar is a palindrome

Enter string to check:
>>>
```

Task: In IDLE, open a new project and save at as Palindrome_yourLastName. Copy the code from the sample on the next pages. Test and revise the program, as needed.

```
1  # Palindrome Checker Program
2
3  import stack
4
5  # welcome
6  print('This program can determine if a given string is a palindrome\n')
7  print('(Enter return to exit)')
8
9  # init
10 char_stack = stack.getStack()
11 empty_string = ''
12
13 # get string from user
14 chars = input('Enter string to check: ')
15
16 while chars != empty_string:
17     if len(chars) == 1:
18         print('A one letter word is by definition a palindrome\n')
19     else:
20         # init
21         is_palindrome = True
22
23         # to handle strings of odd length
24         compare_length = len(chars) // 2
25
26         # push second half of input string on stack
27         for k in range(compare_length, len(chars)):
28             stack.push(char_stack, chars[k])
29
30         # pop chars and compare to first half of string
31         k = 0
32         while k < compare_length and is_palindrome:
33             ch = stack.pop(char_stack)
34             if chars[k].lower() != ch.lower():
35                 is_palindrome = False
36
37             k = k + 1
38
39         # display results
40         if is_palindrome:
41             print(chars, 'is a palindrome\n')
42         else:
43             print(chars, 'is NOT a palindrome\n')
44
45     # get next string from user
46     chars = input('Enter string to check: ')
```

Notes:

The stack module is imported on line 3 of the program. The "`import` *`modulename`*" form of im- port is used. Therefore, each stack function is referenced by `stack.`*`function_name`*. Lines 6–7 displays a simple program welcome. The following lines perform the required initialization for the program. Line 10 sets `char_stack` to a new empty stack by call to `getStack()`. Line 11 initializes variable `empty_string` to the empty string. This is used in the program for determining if the user has finished entering all the words to check.

The string to check is input by the user on line 14. If the string is of length one, then by definition the string is a palindrome. This special case is handled in lines 17–18. Otherwise, the complete string is checked. First, variable `palindrome` is initialized to `True`. On line 24, variable

`compare_ length` is set to half the length of the input string, using integer division to truncate the length to an equal number of characters. This represents the number of characters from the front of the string (working forward) that must match the number of characters on the rear of the string (working backwards). If there are an odd number of characters, then the middle character has no other character to match against.

On lines 27–28 the second half of the string chars are pushed character-by-character onto the stack. Then, on lines 31–37 the characters are popped from the stack one by one, returning in the reverse order that they were pushed. Thus, the first character popped (the last character pushed on the stack) is compared to the first character of the complete string. This continues until there are no more characters to be checked. If characters are found that do not match, then `is_palindrome` is set to `False` (lines 34–35) and the while loop terminates. Otherwise, `is_palindrome` remains `True`. Lines 40–43 output whether the input string is a palindrome or not, based on the final value of `is_palindrome`. Lines 45–46 prompt the user for another string to enter, and control returns to the top of the while loop.

It is important to mention that the problem of palindrome checking could be done more efficiently without the use of a stack. A for loop can be used that compares the characters k locations from each end of the given string. Thus, our use of a stack for this problem was for demonstration purposes only. We leave the checking of palindromes by iteration as a chapter exercise.