Parenthesis Checker

[stack](http://www.practice.geeksforgeeks.org/tag-page.php?tag=stack&isCmp=0) [string](http://www.practice.geeksforgeeks.org/tag-page.php?tag=string&isCmp=0)

[Amazon](http://www.practice.geeksforgeeks.org/tag-page.php?tag=Amazon&isCmp=1) [Snapdeal](http://www.practice.geeksforgeeks.org/tag-page.php?tag=Snapdeal&isCmp=1)

Given an expression string exp, examine whether the pairs and the orders of “{“,”}”,”(“,”)”,”[“,”]” are correct in exp.

For example, the program should print 'balanced' for exp = “[()]{}{[()()]()}” and 'not balanced' for exp = “[(])”

Input:

The first line of input contains an integer T denoting the number of test cases.

Each test case consist of a string of expression, in a separate line.

Output:

Print 'balanced' without quotes if pair of parenthesis are balanced else print 'not balanced' in a separate line.

Constraints:

1 ≤ T ≤ 30

1 ≤ |s| ≤ 100

Example:

Input:

3

{([])}

()

()[]

Output:

balanced

balanced

balanced

#### **\*\*For More Examples Use Expected Output\*\***

**Algorithm:**

1) Declare a character stack S.

2) Now traverse the expression string exp.

a) If the current character is a starting bracket (‘(‘ or ‘{‘ or ‘[‘) then push it to stack.

b) If the current character is a closing bracket (‘)’ or ‘}’ or ‘]’) then pop from stack and if the popped character is the matching starting bracket then fine else parenthesis are not balanced.

3) After complete traversal, if there is some starting bracket left in stack then “not balanced”

Algorithm:

1. i := 0
2. Find a matching pair from i. If none is found, then the string is not valid. If one is found, let i be the index of the first character.
3. Remove [i:i+1] from the string
4. If i is at the end of the string, and the string is not empty, it's a failure.
5. If [i-1:i] is a matching pair, i := i-1 and back to 3.
6. Else, back to 1.

The algorithm is O(n) in complexity because:

* each iteration of the loop removes 2 characters from the string
* the step 2., which is linear, is naturally bound (i cannot grow indefinitely)

And it's O(1) in space because only the index is required.

Of course, if you can't afford to destroy the string, then you'll have to copy it, and that's O(n) in space so no real benefit there!

Unless, of course, I am deeply mistaken somewhere... and perhaps someone could use the original idea (there is a pair somewhere) to better effect.