If Brackets are balanced?

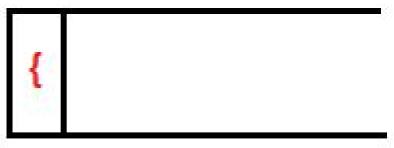
Problem Description: You are given with a string expression, write a program to find whether brackets are balanced or not, i.e. if a bracket opens last, then it should get closed first. Print true if the brackets are balanced, otherwise print false.
For example: { a + [b+ (c + d)] + (e + f) }
Here, all the brackets are balanced, so the output for this input string must be true.

Algorithm to approach:

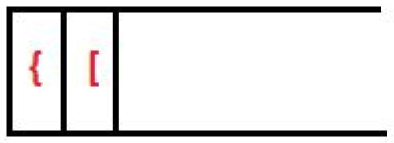
1. You need to declare a Stack S.
2. Now, traverse the string expression.
   - If the current character of the string is an opening bracket(‘(‘ or ‘{‘ or ‘[‘), then push it to the stack.
   - If the current character is a closing bracket (‘)’ or ‘}’ or ‘]’) then pop from stack and if the popped character is the matching opening bracket then fine, else parentheses are not balanced. This is because of the fact that in a balanced brackets expression, whichever bracket opens last, should be closed first.
   - After complete traversal, if there is some starting bracket left in stack then the expression is “not balanced” because this means there are some extra opening brackets.

Let us take an example:
{ a + [b+ (c + d)] + (e + f) }

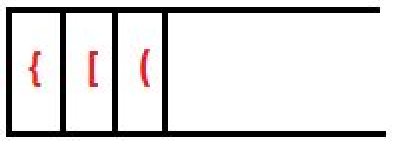
Make a stack S.
We have an opening bracket first, insert it into the stack.



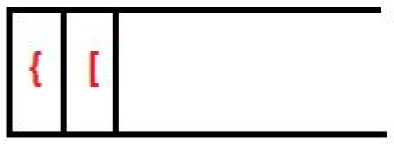
Again an opening bracket, insert into the stack.



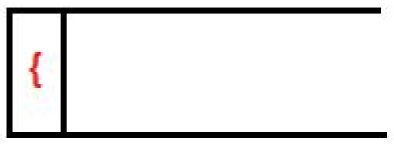
Again, an opening bracket, insert into the stack.



Now, we encounter a closing bracket, pop the topmost element from the stack and it matches the closing bracket. Hence, we continue the process.



Again, we encounter a closing bracket, pop the topmost element from the stack and it matches the closing bracket. Hence, we continue the process.



Again, we encounter a closing bracket, pop the topmost element from the stack and it matches the closing bracket. Hence, we continue the process.

Now, the string finished and we have an empty stack left, hence the brackets are balanced.

Pseudo Code:

```
Function: checkBalanced(exp) {
        Create a Stack  s
        For i=0 to i less than length of exp:
                if(exp[i] = '(' or  exp[i] = '{' or  exp[i] = '[') :
                        push(exp[i]) on stack s

                else if(exp[i] = ')' or  exp[i] = '}' or  exp[i] = ']') :
                        if(stack is empty) :
                                return false

                        ch = top of stack
                        Pop from stack
                        if(exp[i] = ')' and ch = '(') :
                                Do nothing and continue

                        else if(exp[i] = '}' and ch = '{') :
                                Do nothing and continue

                        else if(exp[i] = ']' and ch = '[') :
                                Do nothing and continue

                        else :
                                return false

        if(stack is empty) :
                return true
        else :
                return false
```