HARDIK KHARE | 70765344

- **Identify at least 1 edge case(s)** for the problem and expected results
    1. There are no bracket characters
    2. There are only opening or closing brackets
    3. There are no closing brackets
    4. There are no opening brackets

- **Describe 3 test case**(s) and expected result(s) for the program

1. Input string is valid

   Input: `(does{[[well!]]}work)this`

   Output:

```
this
  does work
        Well!
```

2. Brackets are mismatched

   Input: `(doesn't{[[work!]})this`

   Output:

```
mismatched groups!
```

3. Closing brackets are missing

   Input: `(abc(((cartoon`

   Output:

```
mismatched groups!
```

- **Describe** your algorithm in words
    1. Program starts by taking the text string as input. We define a bracketMap with keys as closing brackets and values being corresponding opening brackets. A List of List structure is defined where each list corresponds to strings at a depth level. Depth is defined as the number of opening/closing brackets.
    2. Function addStringtoDepthList is defined.
        a. This checks if parent list size is at least as much as depth. If not, we add depth number of lists (each corresponding to a depth level) to the parent list.
        b. Then we retrieve the list of strings corresponding to depth and add the current string to this list.
    3. In the parentheses decode function, we iterate over each character:
        a. If a character is one of the opening brackets ‘[‘,’{‘ or ‘(‘, we call the openingBrace function.
            i. In this, call addStringToDepthList function
            ii. Add opening brace to stack
            iii. Increment depth and reset the current string
        b. else If a character is one of the closing brackets ‘]‘,’}‘ or ‘)‘, we call the closingBrace function.
            i. If stack of brackets is empty or the topmost bracket doesn’t correspond to the closing bracket, return -1;
            ii. Call addStringToDepthList function
            iii. Decrement depth and reset current string
        c. Otherwise append current character to string
    4. Step 3 is repeated until we read all characters in input or anytime closingBrace returns -1 (signifying mismatched group), break out of loop and return -1
    5. At the end, if bracket stack is empty, return 0 (i.e valid) or -1 (i.e. mismatched group)
    6. Finally we call displayOutput function where we pass our list of list and validStatus (0 or 1 received from parentheses decode function)

a. If valid status is -1, print ‘mismatched group’.
b. Else print each list level by level. For each level, print 2*level number of whitespaces initially and then all strings at that level.

- **Explain** your algorithm (provide a proof sketch of why it works)

  Aim of this program is to:

  - Validate if parentheses are balanced
  - Print text in the order of nesting

  To validate if parentheses are balanced,

  - Everytime a closing bracket is encountered, we check if the opening bracket stored at the top of the stack is corresponding to the current closing bracket. If it does not correspond or the stack is empty we print ‘mismatched group!’ and return

  To print text in order of nesting

  - Whenever an opening or closing bracket is encountered, we increment or decrement the nesting level variable.
  - Every word/string is associated with a nesting level and stored in a list of lists where each list is for a nesting.
  - While printing the output, we visit the list at every level and print the strings stored in them.

- **Analyze** the **time** and space **complexity** of the solution. Explain why the time and space complexity fit your algorithm.

  We will discuss space and time complexity for each function in our program.

  Overall time complexity is O(n).

  - We are iterating over each character in string just once
  - To add a word/string to the output list, we add it to the list at ‘depth’ level. This takes O(1) time as list retrieval is done by depth index.
  - For printing, we iterate over each word/string only once which takes O(n) time

  Overall space complexity O(input_length)

  - We store opening brackets in a stack which will be O(n) at max when string only consists on opening brackets
  - We don't store closing brackets
  - Each word/string in input is stored only once in a list of lists. This will be O(n) space

- **Test**, and **demonstrate** correct function of the solution (add screenshots of your test cases)

```
hkhare@circinus-29 20:33:16 ~/lab/lab3
$ javac -Xlint APLab3.java
hkhare@circinus-29 20:33:29 ~/lab/lab3
$ java APLab3
Please enter input:
(does{[[well!]]}work)this

this
  does work
        well!
hkhare@circinus-29 20:33:37 ~/lab/lab3
$ java APLab3
Please enter input:
(doesn't{[[work!]]})this
mismatched groups!
hkhare@circinus-29 20:33:55 ~/lab/lab3
$ java APLab3
Please enter input:
Hell(is{all[are]}empty{the}(devils((here)))and)

Hell
  is empty and
    all the devils
      are
        here
hkhare@circinus-29 20:34:07 ~/lab/lab3
$ java APLab3
Please enter input:
ThisString has no brackets

ThisString has no brackets
hkhare@circinus-29 20:34:26 ~/lab/lab3
$ java APLab3
Please enter input:
{{{{{
mismatched groups!
hkhare@circinus-29 20:34:35 ~/lab/lab3
$ java APLab3
Please enter input:
]]]]}
mismatched groups!
hkhare@circinus-29 20:34:40 ~/lab/lab3
```