**Problem #1 (Marks: 5)**

**Stack**

A *stack* is a data structure that is built using an array. When you add a new element to stack, it gets added at the top of the stack. This operation is called a *push*. You can read or extract only the top element of the stack at any time. This operation is called a *pop*. Essentially, the last element entered in the stack gets extracted first. Hence it is called a LIFO (last in, first out) data structure. (Imagine getting in and out of a lift! The last person entering the lift is the first person to get out.) Your task is to implement a stack of characters using a global array (called *stack*) and another global integer variable (called *tos*) that indexes the top of the stack. Use a defined constant SIZE (set to 50) to specify the size of the stack array. Implement the following functions:



* push – Adds a character at the top of stack
* pop – Removes and returns the top character from the stack
* isEmpty – returns TRUE (non-zero) if the stack is empty. If non-empty, then it returns FALSE (zero).

**Problem #2 (Marks: 5)**

**Brackets Matching**

In this problem, you will use the stack data structure that you have implemented. Using it, you need to find whether an input string of brackets is error free or not. That means, each opening bracket must have a matching closing bracket of same type and there wouldn’t be any extra unmatched closing brackets. Here is how you can do this:

Start with an empty stack. If the current character is an opening bracket, just push it to the stack. If it's a closing bracket, check that the stack is not empty and pop operation returns an appropriate opening bracket (that it is, matches this one). If the stack was empty or the top of stack didn’t have matching type of opening bracket, then report error at the current position of the string. Otherwise repeat this process until the input string is completely processed. Finally, verify that the stack is empty. If the stack isn’t empty at this point, then report error at the end of the string.

The input contains a string consisting of symbols from the set {‘(‘, ‘{‘, ‘[‘,‘)‘, ‘}‘, ‘]‘}.

|  |  |
| --- | --- |
| **Sample Input(s)** | **Corresponding Output(s)** |
| ({}(([]{})[]){()})  ({}(([]{)}[]){()})  ({}(([] | Matched  Error at position 9  Error at position 8 |