5TH JAN 2023

FIRST :-

Parenthesis Checker :: Easy

Given an expression string **x**. Examine whether the pairs and the orders of “{”,”}”,”(”,”)”,”[”,”]” are correct in exp.
For example, the function should return 'true' for exp = “[()]{}{[()()]()}” and 'false' for exp = “[(])”.

Example 1:

```
Input:
{([])}
Output:
true
Explanation:
{ ( [ ] ) }. Same colored brackets can form
balaced pairs, with 0 number of
unbalanced bracket.
```

Example 2:

```
Input:
()
Output:
true
Explanation:
(). Same bracket can form balanced pairs,
and here only 1 type of bracket is
present and in balanced way.
```

Example 3:

```
Input:
([]
Output:
false
Explanation:
([]. Here square bracket is balanced but
the small bracket is not balanced and
Hence , the output will be unbalanced.
```

Your Task:
This is a **function** problem. You only need to complete the function **ispar()** that takes a **string** as a **parameter** and returns a boolean value **true** if **brackets** are **balanced** else **returns false**. The **printing** is done **automatically** by the **driver code**.

Expected Time Complexity: O(|x|)
Expected Auixilliary Space: O(|x|)

Constraints:
1 ≤ |x| ≤ 32000

Note: The drive code prints "balanced" if function return true, otherwise it prints "not balanced".

HINT SECTION:-

1. Create a stck of character.
2. Traverse the whole string
   a. If any char is opening of parenthesis (or { or [,then push it into stack
   b. Else if it is an closing bracket if yes then check it for top of the stack is containing the same or not,if it same as the top of the stack then pop it and move to next
   c. Else return false
3. After finishing the traversal if stack is not empty that means it's not balanced….
4. Else it is balanced.

CODE SECTION:-

```
bool ispar(string s)
   {
       // Your code here
       stack<char> st;
       int i = 0;
       while (s[i])
       {
           if (s[i] == '(' || s[i] == '{' || s[i] == '[')
           {
               st.push(s[i]);
               i++;
           }
           else if (s[i] == ')' || s[i] == '}' || s[i] == ']')
           {
               if (st.empty())
               {
                   return false;
               }
               if (s[i] == ')' && st.top() != '(')
               {
                   return false;
               }
               if (s[i] == '}' && st.top() != '{')
               {
                   return false;
               }
               if (s[i] == ']' && st.top() != '[')
               {
                   return false;
               }

               st.pop();
               i++;
           }
           else
           {
               i++;
           }
       }

       return st.empty();
}
```

SECOND:-

Get minimum element from stack :: Medium

You are given **N** elements and your task is to Implement a Stack in which you can get minimum element in O(1) time.

Example 1:

```
Input:
push(2)
push(3)
pop()
getMin()
push(1)
getMin()
Output: 2 1
Explanation: In the first test case for
query
push(2)  Insert 2 into the stack.
         The stack will be {2}
push(3)  Insert 3 into the stack.
         The stack will be {2 3}
pop()    Remove top element from stack
         Poped element will be 3 the
         stack will be {2}
getMin() Return the minimum element
         min element will be 2
push(1)  Insert 1 into the stack.
         The stack will be {2 1}
getMin() Return the minimum element
         min element will be 1
```

Your Task:
You are required to complete the three methods **push()** which take one argument an integer **'x'** to be pushed into the stack, **pop()** which returns a integer poped out from the stack and **getMin()** which returns the min element from the stack. (-1 will be returned if for **pop() and getMin()** the stack is empty.)

Expected Time Complexity : O(1) for all the 3 methods.
Expected Auixilliary Space : O(1) for all the 3 methods.

Constraints:
1 <= Number of queries <= 100
1 <= values of the stack <= 100

HELP SECTION:-

- CREATE ANOTHER STACK
- **IN PUSH FUNCTION**
  - Push the x to the both if our new stack is empty
  - Else push it to old stack and check if the value of top of the old stack I lower than the value of top of the new stack then push the *value to the new stack also…*
- **In pop**
  - Check if our new stack is empty then return -1;
  - Else check the top of old and new stack are equal then pop from new.
  - Store the value of old top in a variable and return it(this is your answer)
- **In getmin**
  - if our new stack is empty then return -1;
  - else return the top of our new stack coz this stack is containing the minimum of the element only.

CODE SECTION:-

```
class Solution
{
    int minEle;
    stack<int> s;
    stack<int> as;
public:
    /*returns min element from stack*/
    int getMin()
    {
        // Write your code here
        if (as.empty() == true)
            return -1;
        return as.top();
    }
    /*returns poped element from stack*/
    int pop()
    {
        if (as.empty() == true)
        {
            return -1;
        }
        if (s.top() == as.top())
        {
            as.pop();
        }
        int ans = s.top();
        s.pop();
        return ans;
    }
    /*push element x into the stack*/
    void push(int x)
    {

        if (as.empty() == true)
        {
            s.push(x);
            as.push(x);
        }
        else
        {
            s.push(x);
            if (as.top() >= s.top())
            {
                as.push(x);
            }
        }
    }
};
```

-: Done for the today :-