Sample Problems on stacks

Problem #1 : Print Reverse of linked List using Stack.

Description - We are given a linked list. We have to print the reverse of the linked list using Stack.

```
[1 2 3 4 5]
[5 4 3 2 1]
```

Solution - We will traverse the linked list and push all the nodes of the linked list to the stack. Since stack have property of Last In, First Out, List is automatically reversed when we will pop the stack elements one by one.

Pseudo Code

```
void printreverse(Node *head)
{
    stack < Node* > s
    current = head
    while(current != NULL)
    {
        s.push(current)
        current = current->next
    }
    while( ! s.empty())
    {
        node = s.top()
        print(node->data)
        s.pop()
    }
}
```

Time Complexity : O(n)
Auxiliary Space : O(n)

Problem #2 : Check for balanced parentheses in an expression

Description - Given an expression string exp , we have to check whether the pairs and the orders of **{ " , " }** , **(" , ")** and **[" , "]** are correct in exp. For example -

```
Input : [ ( ) ] { } { [ ( ) ( ) ] ( ) }
Output : true
Input : [ ( ] )
Output : false
```

Solution : Follow the streps below -

1. Declare a character stack S.
2. Now traverse the expression string exp.
    - If the current character is a starting bracket '(' or '{' or '[' then push it to stack.
    - 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".

Pseudo Code

```
// function to check if paranthesis are balanced
bool areParanthesisBalanced(string expr)
{
    stack < char > s

    for i=0 to expr.size() {

        if (expr[i]=='('||expr[i]=='['||expr[i]=='{') {
            s.push(expr[i])
            continue
        }
```

```
        // stack can not be empty for closing bracket
        if s.empty()
            return false

        switch (expr[i]) {
            case ')':  {
                x = s.top()
                s.pop()
                if (x=='{' || x=='[')
                    return false
                break
            }
            case '}':  {
                x = s.top();
                s.pop();
                if (x=='(' || x=='[')
                    return false
                break
            }
            case ']':  {
                x = s.top();
                s.pop();
                if (x =='(' || x == '{')
                    return false
                break
            }
        }
    }
    // Check Empty Stack
    return (s.empty())
}
```

Time Complexity : O(n)
Auxiliary Space : O(n)

Problem #3 : Next Greater Element

Description : Given an array, print the Next Greater Element (NGE) for every element. The Next greater Element for an element x is the first greater element on the right side of x in array. Elements for which no greater element exist, consider next greater element as -1.

```
Input : [13, 7, 6, 12]
        Output
 Element       NGE
   13   -->     -1
   7    -->     12
   6    -->     12
   12   -->     -1
```

Solution : Follow the below steps -

1. Push the first element to stack.
2. Pick rest of the elements one by one and follow the following steps in the loop.
    1. Mark the current element as next.
    2. If the stack is not empty, compare top element of the stack with next.
    3. If next is greater than the top element, Pop element from the stack. next is the next greater element for the popped element.
    4. Keep popping from the stack while the popped element is smaller than next. next becomes the next greater element for all such popped elements
    5. Finally, push the next in the stack.
3. After the loop in step 2 is over, pop all the elements from stack and print -1 as the next element for them.

Pseudo Code

```
// Next greater Element
void printNGE(arr, n)
{
    stack < int > s
    s.push(arr[0])
    for i=0 to n-1 {
        if (s.empty()) {
            s.push(arr[i])
            continue
        }
        while (s.empty() == false && s.top() < arr[i]) {
            print(s.top() + "-->" + arr[i])
            s.pop()
        }
        s.push(arr[i])
        while (s.empty() == false) {
           print(s.top() + "-->" + -1)
           s.pop()
        }
    }
}
```

Time Complexity : O(n)
Auxiliary Space : O(n)