Stack Fundamentals

Intro To Stacks
Implementation using array
Implementation Using LL
Balanced Parenthesis
Remove equal pair of consecutive characters
Evaluates postfix expressions

Intro To Stacks

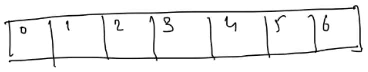
Stacks are a type of container adaptors with LIFO(Last In First Out) type of working, where a new element is added at one end (top) and an element is removed from that end only. Stack uses an encapsulated object of either vector or deque (by default) or list (sequential container class) as its underlying container, providing a specific set of member functions to access its elements.

Inbuilt functionalities of Stacks in C++ STLs / Operations

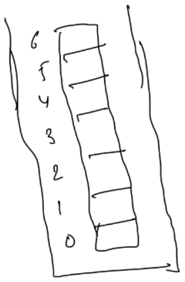
empty() – Returns whether the stack is empty – Time Complexity : O(1)
size() – Returns the size of the stack – Time Complexity : O(1)
top() – Returns a reference to the top most element of the stack – Time Complexity : O(1)
push(g) – Adds the element ‘g’ at the top of the stack – Time Complexity : O(1)
pop() – Deletes the most recent entered element of the stack – Time Complexity : O(1)

Implementation using array

```
int A[7]
int t = -1; //point to top
```



Capacity | limit = 7



```
void push (int data)
{ if (t = len -1) {
      return;
  }        //stack overflow
  t++;
  A[t] = data;
}
```

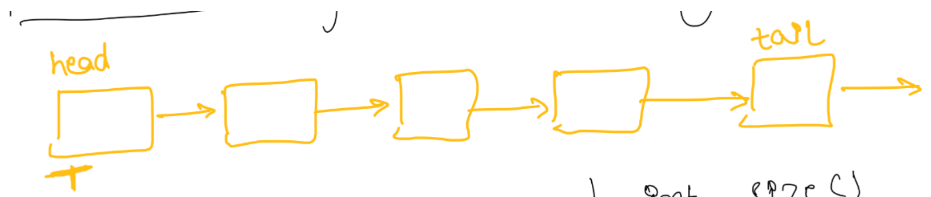
```
int top()
{ if (isEmpty()) {
      return;
  }
  return A[top];
}
```

```
void pop()
{ if (isEmpty())
  {
     return;
  }  //stack underflow
  t--;
}
```

```
bool isEmpty()
{
    return (t == -1)
}
```

T.C = O(1)

Implementation Using LL



```
void push(int data)
{
  Node nn = new Node(data);
  nn.next = head;
  head = nn
  cnt = cnt+1;
}

void pop()
{ if (isEmpty())
  {
     return;
  }
  head = head.next;
  cnt = cnt-1;
}
```

$\boxed{T.C \forall = O(1)}$

```
int size()
{
  return cnt;
}

int top()
{ if (isEmpty())
       return;
  return head.data;
}

bool isEmpty()
{
 return (head == NULL);
}
```

Balanced Parenthesis

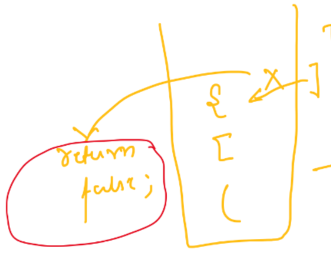
Q Check whether given sequence of parenthesis is valid or not. ?

eg:- (({})) → valid

([{]) → Invalid

→ for every closing parenthesis, the most recent opening should be of the same type.

Top of stack = most recent opening



→ If st == empty everything has matched ⇒ valid parenthesis

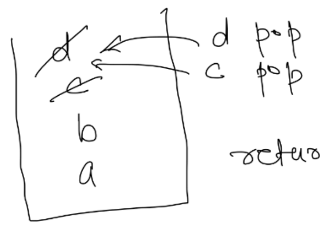
Code :-

```
for char : string :
    if 'char' is in ('(' || '{' || '[')
    |       push 'char' in Stack;
    |
    else if closing parenthesis :
    |       if (st.isempty())
    |           return false;
    |       if top is not of the same type of 'char'
    |           return false;

if (st.isempty)
|       return true;
else
|       return false

                                T.C : O(N)
                                S.C : O(N)
```

Remove equal pair of consecutive characters

Q⇒ Given a string, remove equal pair of adjacent characters till you can.

Eg:- abcdde
ab cc
ab



return "a,b";

Code:-

```
for 'char' : string :
    if (!st.empty() && 'char' == st.top)
            st.pop();
    else
        st.push('char');

while (!st.empty)
    //add each element of stack to string;

return string;
```

T.C : O(N)
S.C : O(N)

Evaluates postfix expressions

Q3 Given a postfix expression, evaluate it.

2 + 3 : Infix expression

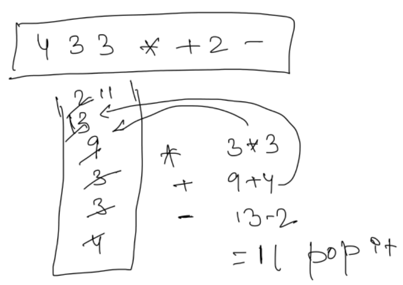
Operator

operands

+2 3 → prefix

23 + → postfix.

eg:



Code:

```
for ele in expression :
    if ele is operand :
        st.push (ele)
    else
        pop (operand1)
        pop (operand2)
    ans = Calc (operand1, operand2, ele) ;
        push ( ans )

if ( st. size == 1)
    return st.top() ;
```