

**AIM: Implementation of Stack Applications like: Postfix evaluation & Balancing of Parenthesis**

**THEORY:**

The stack is a linear data structure which follows the last in first out (LIFO) principle. Applications of stack:

- 1) Balancing of symbols.
- 2) Infix to postfix /Prefix conversion
- 3) Redo-undo features at many places like editors, photoshop.
- 4) Forward and backward feature in web browsers
- 5) Used in many algorithms like Tower of Hanoi, tree traversals, stock span problem, histogram problem.
- 6) Backtracking is one of the algorithm designing technique .Some example of backtracking are Knight-Tour problem, N-Queen problem, find your way through maze and game like chess or checkers in all this problems we dive into someway if that way is not efficient we come back to the previous state and go into some other path. To get back from the current state we need to store the previous state for that purpose we need stack.
- 7) In Graph Algorithms like topological sorting and Strongly Connected Components.
- 8) In Memory management any modern computer uses stack as the primary-management for a running purpose. Each program that is running in a computer system has its own memory allocations

- **POSTFIX EVALUATION:**

The Postfix notation is used to represent algebraic expressions. The expressions written in postfix form are evaluated faster compared to infix notation as parenthesis are not required in postfix.

Following is algorithm for evaluation postfix expressions.

- 1) Create a stack to store operands (or values).
- 2) Scan the given expression and do following for every scanned element.
  - a. If the element is a number, push it into the stack
  - b. If the element is an operator, pop operands for the operator from stack. Evaluate the operator and push the result back to the stack.
- 3) When the expression is ended, the number in the stack is the final answer.

- **BALANCING OF PARENTHESIS:**

Steps to find whether a given expression is balanced or unbalanced:

- 1) Input the expression and put it in a character stack.
- 2) Scan the characters from the expression one by one.
- 3) If the scanned character is a starting bracket ( ' ( ' or ' { ' or ' [ ' ), then push it to the stack.
- 4) If the scanned character is a closing bracket ( ' ) ' or ' } ' or ' ] ' ), then pop from the stack and if the popped character is the equivalent starting bracket, then proceed. Else, the expression is unbalanced.
- 5) After scanning all the characters from the expression, if there is any parenthesis found in the stack or if the stack is not empty, then the expression is unbalanced.

**A) POSTFIX EVALUATION:**

**SOURCE CODE:**

```
#include <iostream>
#include <string>
#include <stack>
using namespace std;

int evalPostfix(string exp)
{
    stack<int> stack;

    for (char c: exp)
    {
        if (c >= '0' && c <= '9') {
            stack.push(c - '0');
        }
        else {
            int x = stack.top();
            stack.pop();

            int y = stack.top();
            stack.pop();

            if (c == '+') {
                stack.push(y + x);
            }
            else if (c == '-') {
                stack.push(y - x);
            }
            else if (c == '*') {
                stack.push(y * x);
            }
            else if (c == '/') {
                stack.push(y / x);
            }
        }
    }

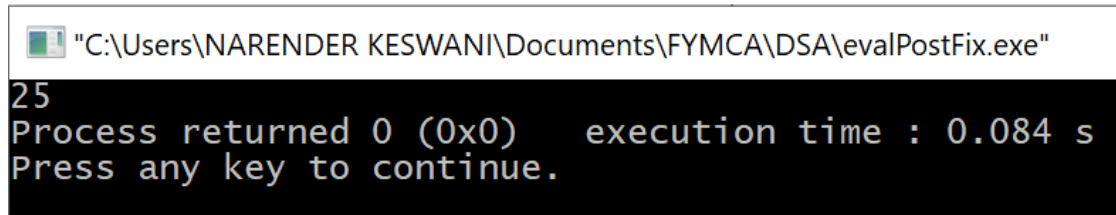
    return stack.top();
}

int main()
{
    string exp = "138*+";

    cout << evalPostfix(exp);

    return 0;
}
```

OUTPUT:



**B) BALANCED PARENTHESIS:**

SOURCE CODE:

```
#include <bits/stdc++.h>
using namespace std;

bool checkBalancing(string expr)
{
    stack<char> s;
    char x;

    for (int i = 0; i < expr.length(); i++)
    {
        if (expr[i] == '(' || expr[i] == '['
            || expr[i] == '{')
        {
            s.push(expr[i]);
            continue;
        }

        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;
    }
}


return (s.empty());
}

int main()
{
    string expr;
    cout<<"Enter a string \n";
    cin>>expr;

    if (checkBalancing(expr))
        cout << "Balanced";
    else
        cout << "Not Balanced";
    return 0;
}
```


**OUTPUT:**

**CASE-I BALANCED:**

 "C:\Users\NARENDER KESWANI\Documents\FYMCA\DSA\balancedParentheses.exe"

```
Enter a string
{[1+2][1]}
Balanced
Process returned 0 (0x0)   execution time : 28.767 s
Press any key to continue.
```

**CASE-II UNBALANCED:**

 "C:\Users\NARENDER KESWANI\Documents\FYMCA\DSA\balancedParentheses.exe"

```
Enter a string
{[15+6}
Not Balanced
Process returned 0 (0x0)   execution time : 8.272 s
Press any key to continue.
```