**TITLE:-** Implementation and Usage of Stack

**Objective:-** Development of arithmetic expressions evaluation system.

**THEORY:-**

• DSA : A data structure Is a named location that is used to store and organize data. And, an algorithm is a collection of steps to solve a particular problem. Learning data structures and algorithms allow us to write optimized and efficient computer programs.

•STACK : Stack is a conceptual structure consisting of a set of homogeneous elements and is based on the principle of last in first out (LIFO). It is a commonly used abstract data type with two major operations, namely push and pop.

**Implementation:**

Source Code:

**#include "iostream"**

**#include "algorithm"**

**#include "math.h"**

**#include "string"**

**using namespace std;**

**//Creating Stack**

**template<typename T>**

**class Stack{**

**T \*arr;**

**int nextIndex,capacity;**

**public:**

**Stack(){ //declaring variable**

**capacity = 5;**

**arr = new T[capacity];**

**nextIndex = 0;**

**}**

**int size(){**

**return nextIndex; //return nextIndex that assign size of stack**

**}**

**bool isEmpty() //check is stack empty**

**{**

**if(nextIndex==0){**

**return true;**

**} else return false;**

**}**

**void push(T element){ //define push Method**

**if(nextIndex==capacity){**

**T \*newArr = new T[2\*capacity];**

**for (int i = 0; i < capacity; i++) {**

**newArr[i]=arr[i];**

**}**

**delete []arr;**

**}**

**arr[nextIndex]=element;**

**nextIndex++;**

**capacity=2\*capacity;**

**}**

**void pop(){ //Define POP Method**

**if(isEmpty()) {**

**cout << "Stack is Empty" << endl;**

**delete []arr;**

**return;**

**}**

**nextIndex--;**

**}**

**T top(){**

**if (isEmpty()){**

**cout<<"Stack empty"<<endl;**

**return 0;**

**}**

**return arr[nextIndex-1];**

**}**

**void checkArrayValue(){**

**for(int i=0;i<capacity;i++){**

**cout<<arr[i]<<endl;**

**}**

**}**

**};**

**//Creating class for infix to postfix or prefix and evaluate them**

**class InfixExpression:public Stack<char>{**

**int precendence(char c){**

**if(c=='^'){**

**return 3;**

**}**

**else if(c=='\*'||c=='/'){**

**return 2;**

**}**

**else if(c=='+'||c=='-'){**

**return 1;**

**} else{**

**return -1;**

**}**

**}**

**public:**

**//Creating function to check balanced parenthesis**

**//**

**// string isParenthesisBalanced(string infix){**

**//**

**// }**

**//creating function to convert infix to Postfix**

**string infixToPostfix(string infix){**

**string result;**

**for(int i=0;i<infix.length();i++)**

**{**

**if(infix[i]>='a'&& infix[i]<='z'||infix[i]>='A'&& infix[i]<='Z'){**

**result+=infix[i];**

**}**

**else if(infix[i]=='('){**

**push(infix[i]);**

**}**

**else if(infix[i]==')'){**

**while (!isEmpty()&& top()!='('){**

**result+=top();**

**pop();**

**}**

**if(!isEmpty()){**

**pop();**

**}**

**} else{**

**while(!isEmpty()&& precendence(top())>=precendence(infix[i]))**

**{**

**result+=top();**

**pop();**

**}**

**push(infix[i]);**

**}**

**}**

**while(!isEmpty()){**

**result+=top();**

**pop();**

**}**

**return result;**

**}**

**string infixToPrefix(string infix){**

**reverse(infix.begin(),infix.end());//reverse infix**

**//Replace '(' With ')'**

**for(int i=0;i<infix.length();i++){**

**if(infix[i]=='(')**

**{**

**infix[i]=')';**

**}**

**else if(infix[i]==')')**

**{**

**infix[i]='(';**

**}**

**}**

**string prefix=infixToPostfix(infix);//apply logic of postfix after reverse infix**

**reverse(prefix.begin(),prefix.end());//finally to return infix reverse output**

**return prefix;//return prefix expression**

**}**

**};**

**//create class to evaluate the expression using postfix or prefix;**

**class EvaluateExpression:public Stack<double>{**

**int integerInput[20];**

**public:**

**string userInput(string polishExpression) {**

**char number;**

**string expression = polishExpression;**

**int n = polishExpression.length();**

**char char\_array[n+1];**

**cout<<expression<<endl;**

**for (int i = 0; i < expression.length(); i++) {**

**if (expression[i] >= 'a' && expression[i] <= 'z' || expression[i] >= 'A' && expression[i] <= 'Z') {**

**cout << "Input for " << expression[i] << ": ";**

**cin >> integerInput[i];**

**//expression[i]=number;**

**}**

**}**

**return expression;**

**}**

**double evaluateExpression(string result){**

**//Evalute the UserInput using respective expression**

**for(int i=0;i<result.length();i++)**

**{**

**if (result[i] >= 'a' && result[i] <= 'z' || result[i] >= 'A' && result[i] <= 'Z')**

**{**

**push(integerInput[i]);//if we substract a string to '0' we get integer**

**} else{**

**int operand2 = top();**

**pop();**

**int operand1 = top();**

**pop();**

**switch (result[i]) {**

**case '+':**

**push(operand1+operand2);**

**break;**

**case '-':**

**push(operand1-operand2);**

**break;**

**case '\*':**

**push(operand1\*operand2);**

**break;**

**case '/':**

**push(operand1/operand2);**

**break;**

**case '^':**

**push(pow(operand1,operand2));**

**break;**

**}**

**}**

**}**

**return top();**

**}**

**};**

**int main(){**

**InfixExpression infix;**

**EvaluateExpression evalute;**

**string output=infix.infixToPostfix("x+y\*(z/w)");**

**string postfixOutput=evalute.userInput(output);**

**cout<<evalute.evaluateExpression(postfixOutput);**

**return 0;**

**}**

**DISCUSSSIN AND CONCLUSION:**

**Here,** we got to know the implementation of given real problem by using stack.

Hence, we became familiar with stack by using real world problem