DS-LAB EXPERIMENT 2

NAME: YASH SARANG D6AD/47

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Aim:** To implement an application of stack using Array.

i. Parenthesis Matching.

ii. Infix to Postfix using Stack.

iii. Infix to Prefix using Stack.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Theory:** Stack is a linear data structure that follows a particular order in which the operations are performed.

Few of its usages are given below:

1. Expression Conversion- a. Infix to Postfix

b. Infix to Prefix

c. Postfix to infix

d. Prefix to infix

1. Expression Evaluation
2. Parsing
3. Simulation of Recursion
4. Function Call

There are 3 popular methods for the representation of expressions.

1. Infix x+y (operator between operands)
2. Prefix +xy (operator before operands)
3. Prefix xy+ (operator after operands)

Infix expressions are not used inside a computer, due to the additional complexity of handling precedence.

Prefix and Postfix expressions are free from any precedence.

They are more suited for mechanization.

The computer uses the postfix form for the representation of an expression.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**1. PARENTHESIS MATCHING**

**Algorithms:** s-stack of characters   
 x-character type

Step 1: START

Step 2: x reads the next token

Step 3: if (x==’(‘) Push(s,x);

Step 4: if (x==’)‘) if(top element of stack is ‘(‘)

pop(s);

else print “Mismatch”

Step 5: if (more tokens) - go to - step 1

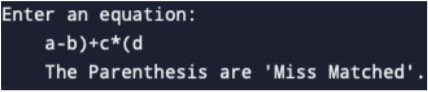
Step 6: if (stack is not empty) Print "Mismatch"

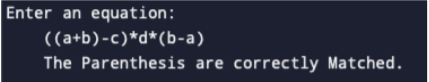
else Print " Proper Paranthesis”

Step 7: Stop.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**OUTPUT:**

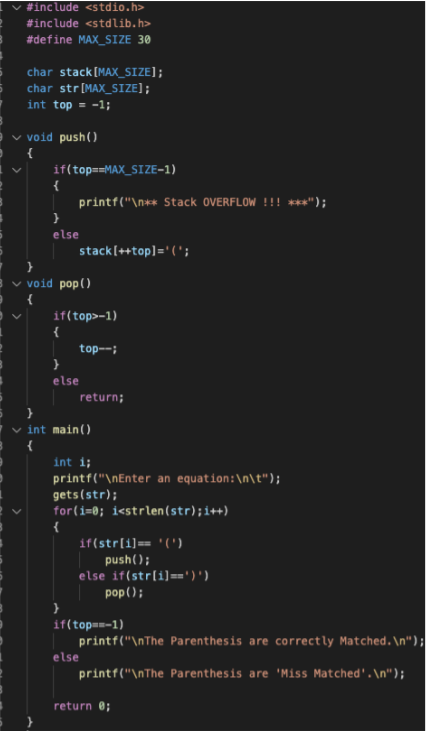




\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**PROGRAM CODE:**

****

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**2. Infix to Postfix using Stack**

**Algorithm:**

Step 1: START

Step 2: Push "("onto Stack, and add ")" to the end of X.

Step 3: Scan X from left to right. Repeat 3-6 for each element of X until the stack is empty.

Step 4: If an operand is encountered, add it to Y.

Step 5: If (opening parenthesis is encountered) push it into Stack.

Step 6: If (operator is encountered),

repeatedly pop from Stack and add each operator to Y (on the top of stack)

which has the same precedence as or higher precedence than the operator.

Add the operator to thestack.

Step 7: If (closing parenthesis is encountered),

repeatedly pop from Stack and add each operator to Y (on the Stack)

until a left parenthesis is encountered.

Remove the left parenthesis.

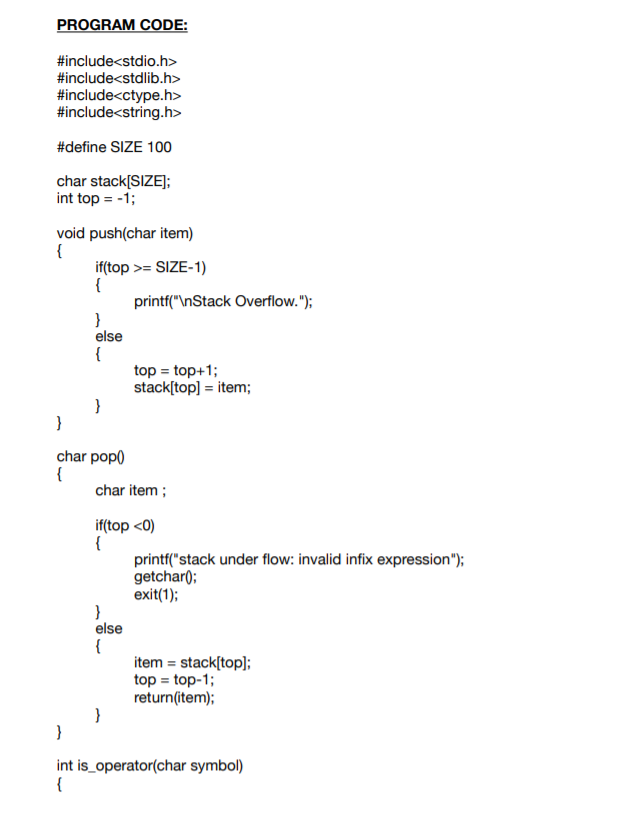
Step 8: Stop

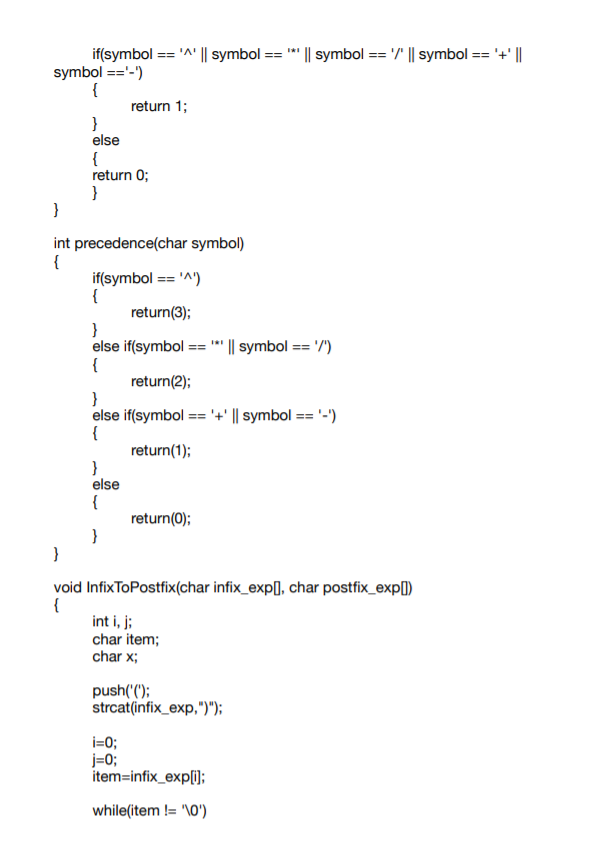
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

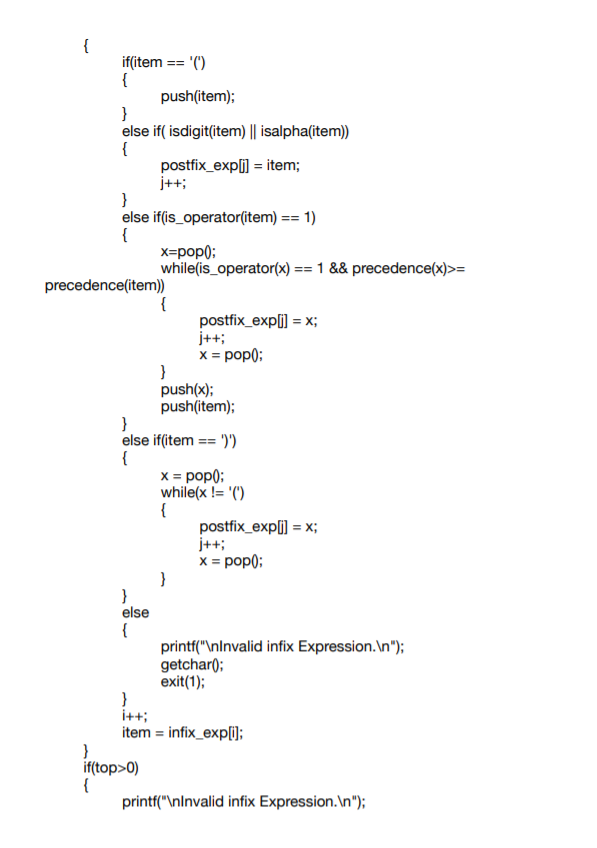
**OUTPUT:**

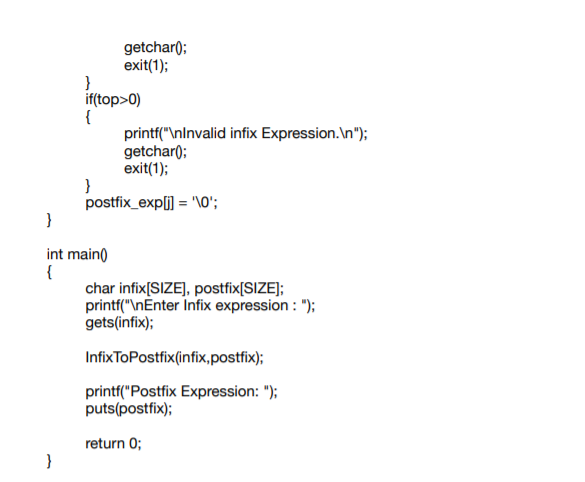


\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_







**** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**3. Infix to Prefix using Stack**

**Algorithm**

Step 1: START

Step 2: Push ")" onto the stack, and add "(" to the end of A.

Step 3: Scan A from right to left and repeat steps 3 to 6 for each element of A

until the Stack is empty.

Step 4: If an operand is encountered add it to B.

Step 5. If a right parenthesis is encountered, push it onto the Stack.

Step 6: If an operand is encountered then:

Repeatedly pop from Stack and add to B each operator

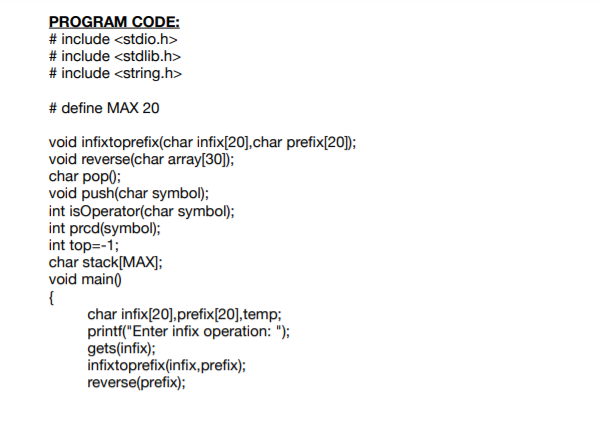
(on the top of STACK) which has the same on higher precedence than the operator. Add operator to STACK.

Step 7: If left parenthesis is encountered then:

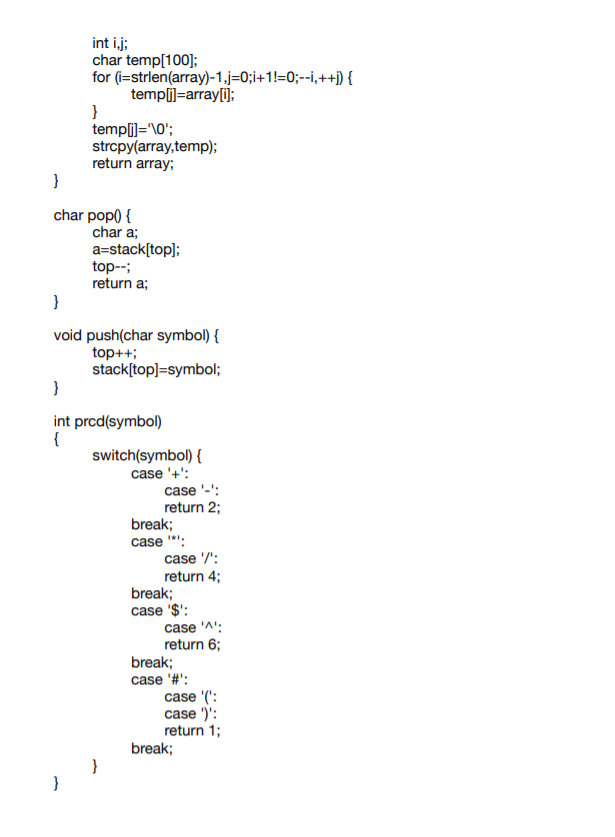
Repeatedly pop from Stack and add to B (each operator on top of Stack until a left parenthesis is encountered)

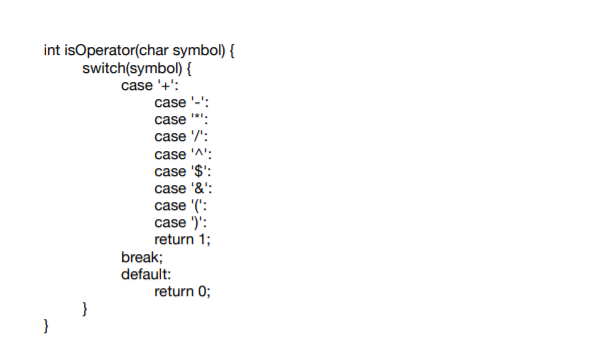
Remove the left parenthesis

Step 8: Stop









\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**OUTPUT:**



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**• Conclusion:**   
 We have learned important elements and applications of Stack using Arrays.   
 We have studied and implemented Parenthesis Matching and Expression

Conversion mainly Infix to Postfix and Infix to Prefix.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_