**Assignment No.: 2**

**Title: Expression Conversion And Evaluation Using Stack ADT**

**Course Outcome:** **CO1(C214447.1):** Analyze algorithms and to determine algorithm correctness and time efficiency class.

**CO2(C214447.2):** Implement abstract data type (ADT) and data structures for given application.

**CO3(C214447.3):** Design algorithms based on techniques like brute -force, divide and conquer, greedy, etc.).

**CO5(C214447.5):** Analyze of algorithms with respect to time and space complexity.

**Date of Completion:**

**Assessment Grade / Marks:**

**Assessor’s Sign with Date:**

**Assignment No: 02**

**Title:** Expression Conversion And Evaluation Using Stack ADT

**Aim:** To study implementation of stack as an abstract data type using singly linked list

**Objective:** To study implementation of stack as an abstract data type using singly linked list

**Problem Statement:** Implement stack as an abstract data type using singly

linked list and use this ADT for Conversion of

• Infix Expression to Postfix,

• Infix Expression to Prefix Evaluation of

• Postfix Expression

• Prefix Expression.

**Course Outcome:** CO Number: Applicable CO : Blooms Taxonomy Category

**Requirements:** (Components / Digital Kits / Platform / Software / Hardware)

**Platform :-** Online GDB Compiler

**Theory / Procedure / Diagrams / Circuits:**

**Linked List:-**

“Linked List” is a linear data structure. Unlike arrays, linked list elements are not stored at a

contiguous location; the elements are linked using pointers.

**Singly Linked List:-**

Singly linked list can be defined as the collection of ordered set of elements. The number of elements may vary according to need of the program. A node in the singly linked list consist of two parts: Data part and Link part.Data part of the node stores actual information that is to be represented by the node while the link part of the node stores the address of its immediate successor.

**Stack :-**

A Stack is a linear data structure that follows the LIFO (Last-In-First-Out) principle. Stack has one end, whereas the Queue has two ends (front and rear). It contains only one pointer top pointer pointing to the topmost element of the stack. Whenever an element is added in the stack, it is added on the top of the stack, and the element can be deleted only from the stack. In other words, a stack can be defined as a container in which insertion and deletion can be done from the one end known as the top of the stack.

**Push Operation :-**

Push operation refers to inserting an element in the stack. Since there’s only one

position at which the new element can be inserted — Top of the stack, the new element is inserted at the top of the stack.

**Pop Operation :-**Pop operation refers to the removal of an element. Again, since we only have access to theelement at the top of the stack, there’s only one element that we can remove. We just remove the top of the stack.

**Infix Expression :-** An infix expression is a single letter, or an operator, proceeded by one infix string and followed by another infix string.

Eg: A+B

**Prefix Expression :-**A prefix expression is a single letter, or an operator, followed by two prefix strings. Every prefix string longer than a single variable contains an operator, first operand and second operand.

Eg:+AB

**Postfix Expression :-**A postfix expression (also called Reverse Polish Notation) is a single letter or an operator, preceded by two postfix strings. Every postfix string longer than a single variable contains first and second operands followed by an operator.

Eg:- AB+

**Algorithm / Methods / Steps: (if applicable)**

MAIN FUNCTION()

STEP 1: Define a 'Node' structure with two members data and next

STEP 2: Define a Node pointer 'top' and set it to NULL.

SUB FUNCTION PUSH():

STEP 1: Create a newNode with given value.

STEP 2: Check whether stack is Empty (top== NULL)

STEP 3: If it is Empty, then set newNode → next = NULL.

STEP 4: If it is Not Empty, then set newNode → next= tail.

STEP 5: Set top = newNode.

SUB FUNCTION POP():

STEP 1: Check whether stack is Empty (top == NULL).

STEP 2: If it is Empty, then display "Stack is Empty!!! Deletion is not possible!!!" and terminate the

function

STEP 3: If it is Not Empty, then define a Node pointer 'temp' and set it to 'tail'

STEP 4: Set 'top = top → prev'.

STEP 5: Delete 'temp'. (free(temp)).

SUB FUNCTION DISPLAY():

STEP 1: Check whether stack is Empty (top == NULL).

STEP 2: If it is Empty, then display 'Stack is Empty!!!' and terminate the function.

STEP 3: If it is Not Empty, then define a Node pointer 'temp' and initialize with tail.

STEP 4: Display 'temp → data --->' and move it to the next node. Repeat the same until temp reaches to

the first node in the stack. (temp → next != NULL).

STEP 5: Display 'temp → data ---> NULL'.

SUB FUNCTION INFIXTOPOSTFIX():

STEP 1: Scan the expression from left to right in the infix expression.

STEP 2: If the leftmost character is an operand, set it as the current output to the Postfix string.

STEP 3: And if the scanned character is the operator and the Stack is empty or contains the '(', ')'

symbol, push the operator into the Stack.

STEP 4: If the scanned operator has higher precedence than the existing precedence operator in the

Stack or if the Stack is empty, put it on the Stack.

STEP 5: If the scanned operator has lower precedence than the existing operator in the Stack, pop all the

Stack operators. After that, push the scanned operator into the Stack.

STEP 6: If the scanned character is a left bracket '(', push it into the Stack.

STEP 7: If we encountered right bracket ')', pop the Stack and print all output string character until '(' is

encountered and discard both the bracket.

STEP 8: Repeat all steps from 2 to 8 until the infix expression is scanned.

STEP 9: Print the Stack output.

STEP 10: Pop and output all characters, including the operator, from the Stack until it is not empty.

SUB FUNCTION INFIXTOPREFIX():

STEP 1: Reverse infix expression & swap ‘(‘ to ‘)’ & ‘)’ to ‘(‘

STEP 2: Scan Expression from Left to Right

STEP 3: Print OPERANDs as the arrive

STEP 4: If OPERATOR arrives & Stack is empty, PUSH to stack

STEP 5: IF incoming OPERATOR has HIGHER precedence than the TOP of the Stack, PUSH it on

stack

STEP 6: IF incoming OPERATOR has EQUAL precedence with TOP of Stack && incoming

OPERATOR is ‘^’, POP & PRINT TOP of Stack. Then test the incoming OPERATOR against the

NEW TOP of stack.

STEP 7: IF incoming OPERATOR has EQUAL precedence with TOP of Stack, PUSH it on Stack.

STEP 8: IF incoming OPERATOR has LOWER precedence than the TOP of the Stack, then POP and

PRINT the TOP. Then test the incoming OPERATOR against the NEW TOP of stack.

STEP 9: IF incoming SYMBOL is ‘(‘ PUSH it onto Stack.

STEP 10: IF incoming SYMBOL is ‘)’ POP the stack & PRINT OPERATORs till ‘(‘ is found or Stack

Empty. POP out that ‘(‘ from stack

STEP 11: IF TOP of stack is ‘(‘ PUSH OPERATOR on Stack

STEP 12: Repeat Step 4 to 11 till expression has character

STEP 13: At the end of Expression, POP & PRINT all OPERATORS from the stack

STEP 14: At the end Reverse output string again.

SUB FUNCTION EVALUATION():

STEP 1: Scan the given expression from left to right.

STEP 2: If operand is encountered, push it onto Stack [End If]

STEP 3: If operator is encountered, Pop two elements

A -> Top element

B-> Next to Top element

Evaluate B operator A

push B operator A onto Stack

STEP 4: Repeat step 3 till all the characters are scanned.

STEP 5: Set Result = Pop

STEP 6: END

**Input: (Test Cases / Data sets / Database Links)**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

MAIN MENU

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| 1. Infix to Prefix with Evaluation |

| 2. Infix to Postfix with Evaluation |

| 3. Exit |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter your choice 1

.........................................................

Enter the infix expression::

a+b\*c/d

After step 1...

Entered infix is...a+b\*c/d

After step 2...d/c\*b+a

After step 3...d/c\*b+a

After step 4...

Prefix is...stack is empty

Prefix expression is +a/\*bcd

Evaluation of Prefix expression

Enter the value of d:1

Enter the value of c:2

Enter the value of b:3

Enter the value of a:4

Value of expression is 10

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

MAIN MENU

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| 1. Infix to Prefix with Evaluation |

| 2. Infix to Postfix with Evaluation |

| 3. Exit |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter your choice 2

.........................................................

Enter the infix expression::

a+b\*c/d

stack is empty

Postfix expression is abcd/\*+

Evaluation of Postfix expression

Enter the value of a:4

Enter the value of b:3

Enter the value of c:2

Enter the value of d:1

Value of expression is 10

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

MAIN MENU

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| 1. Infix to Prefix with Evaluation |

| 2. Infix to Postfix with Evaluation |

| 3. Exit |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter your choice