TITLE: IMPLEMENTATION OF STACK

OBJECTIVE:

* To understand basic data structures and algorithm.
* To push, pop and display the data items in stack.

THEORY:

A stack is a linear data structure that follows the Last-In-First-Out (LIFO) principle. It is similar to a stack of items, where the last item added is the first one to be removed. The stack has two primary operations: "push" to add an item to the top of the stack, and "pop" to remove the topmost item. Additional operations like "peek" (to view the top item without removing it) and "isEmpty" (to check if the stack is empty) are also commonly supported. Stacks are used in various applications, including expression evaluation, function call management, undo-redo operations, and backtracking algorithms.

Postfix and prefix expressions are alternative notations for representing mathematical expressions. In postfix notation, also known as Reverse Polish Notation (RPN), operators are placed after their operands. Prefix notation, also known as Polish Notation, has operators placed before their operands. Both notations can be evaluated using stacks as a data structure.

To evaluate a postfix expression, operands are pushed onto the stack, and when an operator is encountered, the top two operands are popped, the operation is performed, and the result is pushed back onto the stack. The final result is the top element of the stack. Similarly, in prefix notation, operands are pushed onto the stack, and operators are evaluated by popping the top two operands, performing the operation, and pushing the result back onto the stack.

Using stacks simplifies the evaluation process by storing operands and intermediate results. Postfix and prefix notations eliminate the need for parentheses and provide a systematic way to evaluate mathematical expressions. They are commonly used in calculators, programming languages, and mathematical algorithms.

**ALGORITHM: -**

* Algorithm to insert data into a stack (PUSH operation)

1. Set TOP=-1.
2. IF (TOP==MAX-1) PRINT “Stack Overflow” and exit.
3. TOP++
4. Read the data to be inserted.
5. Set Stack[TOP]=data.
6. Goto step 2.
7. EXIT.

* Algorithm to remove data from stack: (POP Operation)

1. IF (TOP==-1) PRINT “Stack Underflow” and exit.
2. data=stack[TOP]
3. PRINT the popped data.
4. TOP--
5. Goto step 2.
6. EXIT.

* Algorithm to evaluate postfix expression:

1. Scan postfix expression from left to right.
2. If scanned character is operand then push it on stack.
3. If an operator(×) is encountered then,

* POP the top two elements of stack where A is the top element and B is the next top element.
* Evaluate B (×) A.
* Place the result back in stack.
* Repeat step 1 to 3 until the end of expression.

1. STOP

* Algorithm for converting infix operation into postfix expression:

1. Create an empty stack to hold operators.

2. Initialize an empty string to store the postfix expression.

3. Scan the infix expression from left to right.

4. If an operand is encountered, append it to the postfix string.

5. If an opening parenthesis is encountered, push it onto the stack.

6. If an operator is encountered, repeatedly pop operators from the stack and append them to the postfix string, as long as:

a) The stack is not empty,

b) The top of the stack is not an opening parenthesis (or has higher precedence than the current operator.

Then, push the current operator onto the stack.

7. If a closing parenthesis is encountered, repeatedly pop operators from the stack and append them to the postfix string until an opening parenthesis is encountered. Discard the opening parenthesis.

8. After scanning the entire infix expression, pop any remaining operators from the stack and append them to the postfix string.

9. The resulting postfix string is the desired postfix expression.