DSA Lab Sheet

II Year / II Part

Faculty: Computer and Electronics

Labsheet #2: Implementation of Stack using array

Objectives:

1. Implementation of stack using array
2. Application of Stack Conversion of Infix to Postfix expression
3. Evaluation of Prefix and Postfix expression, matching parenthesis and reversal of string

Theory:

A stack is a linear data structure that follows the Last In First Out (LIFO) principle. The most common operations are:

1. push() – to insert an element
2. pop() – to remove the top element
3. peek() – to view the top element without removing it

Tasks:

Objective#1

```
#include<stdio.h>
#define SIZE 20
int stack[SIZE];
int top = -1;
void push(int value) {
    if (top < SIZE - 1) {
        top++;
        stack[top] = value;
    } else {
        printf("Stack is full\n");
    }
}
void pop() {
    if (top >= 0) {
        printf("Popped: %d\n", stack[top]);
        top--;
    } else {
        printf("Stack is empty\n");
    }
}
void display() {
    if (top >= 0) {
        printf("Stack: ");
        for (int i = 0; i <= top; i++) {
            printf("%d ", stack[i]);
        }
        printf("\n");
```

```
    } else {
        printf("Stack is empty\n");
    }
}
int main() {
    push(10);
    push(20);
    push(30);
    display();   //Stack: 10 20 30
    pop();       //Popped: 30
    display();   //Stack: 10 20
    return 0;
}
```

Assignment:

1.1 Note the output of the above program and modify the above program to add new functionalities:

- peek (Peek the topmost value of the stack)
- isFull and isEmpty (Check if stack is full or empty)
- count (Count number of items in the stack)
- clear (Clear all stack elements)

Objective#2: Conversion of Infix to Postfix Expression

Algorithm:

1. Initialize an empty stack for operators
2. Initialize an empty list for the postfix expression
3. For each token in the infix expression:
   (a) If the token is an operand, add it to the postfix expression
   (b) If the token is '(', push it onto the stack
   (c) If the token is ')':
      i. While top of stack is not '(', pop from stack and append to postfix
      ii. Pop '(' from the stack
   (d) If the token is an operator (e.g., +, -, *, /):
      i. While the stack is not empty and precedence of token $\leq$ precedence of top of stack:
         - Pop from stack and append to postfix
      ii. Push token onto stack
4. While stack is not empty:
   - Pop from stack and append to postfix

5. Return postfix expression

Assignment:
2.1 Using the algorithm, implement Stack to convert infix expression to postfix expression.

Expected outcomes:

- $A + B \Rightarrow AB+$
- $A + B + C \Rightarrow AB + C+$
- $A + B * C \Rightarrow ABC * +$
- $A + (B - C) * D \Rightarrow ABC - D * +$
- $(A + B) * (C + D) \Rightarrow AB + CD + *$
- $A + ((B + C) * (D + E)) \Rightarrow ABC + DE + *+$

Objective#3

Algorithm to Evaluate Prefix Expression Using Stack:

1. Reverse the given prefix expression.
2. Iterate through the reversed expression:
   (a) If the character is an operand (number), push it onto the stack.
   (b) If the character is an operator ($+, -, *, /, \ldots$), pop two operands from the stack, perform the operation, and push the result back onto the stack.
3. After finishing the iteration, the value remaining in the stack is the result of the evaluation.

Assignment:
2.1 Using the algorithm, implement Stack to evaluate prefix statements.

Expected outcomes:

Prefix Expression	Evaluation Process	Result
+9 * 23	*23 = 6, then 9 + 6 = 15	15
* +234	+23 = 5, then 5 * 4 = 20	20
— * 10 + 623	+62 = 8, *108 = 80, then 80 − 3 = 77	77
+ * 54 * 32	*54 = 20, *32 = 6, then 20 + 6 = 26	26
+1 * 23 * 45	*23 = 6, *45 = 20, then 1 + 6 + 20 = 27	27
* − 53 + 21	−53 = 2, +21 = 3, then 2 * 3 = 6	6