K. J. Somaiya College of Engineering, Mumbai-77
(Autonomous College Affiliated to University of Mumbai)

Batch: B1 **Roll No.: 1711072**

Experiment No. 3

Grade: AA / AB / BB / BC / CC / CD /DD

Title: Stacks using Arrays

Objective: Implementations of Infix to Postfix Transformation and its evaluation program.

Expected Outcome of Experiment:

CO	Outcome
CO1	Explain the different data structures used in problem solving

Books/ Journals/ Websites referred:

Thomas Cormen, Charles Leiserson, Ronald Rivest, Clifford Stein; (CLRS) "Introduction to Algorithms", Third Edition, The MIT Press.

Abstract:-

DATA STRUCTURE:

1. char stack[50] //array to store and manipulate expression with size 50.
2. top = -1 //to keep track of the top of the stack.

OPERATIONS:

1. **Push an element**

   This function is used to push a user defined element of character type into the stack. This function is being used to push operator into the stack for converting infix to postfix and to push operand when evaluating the postfix expression.

2. **Pop an element**

   This function is used to return the top element of the stack. It is used in infix to postfix conversion in precedence clash of operators and used in evaluation of postfix expression to get the top two operands.

3. **Determine precedence**

   This function is simply used to assign priority to the operators so that it can be used for checking operator precedence clash and perform push/pop accordingly.

4. **Evaluate postfix expression**

   This function is used to evaluate the postfix expression obtained from infix expression of user. It returns numeric value for the given postfix expression.

Related Theory: -

- Stack is a linear data structure which follows a particular order in which the operations are performed. The order is LIFO (Last In First Out).
- There are many real life examples of stack:

  Consider the simple example of plates stacked over one another in canteen. The plate which is at the top is the first one to be removed, i.e. the plate which has been placed at the bottommost position remains in the

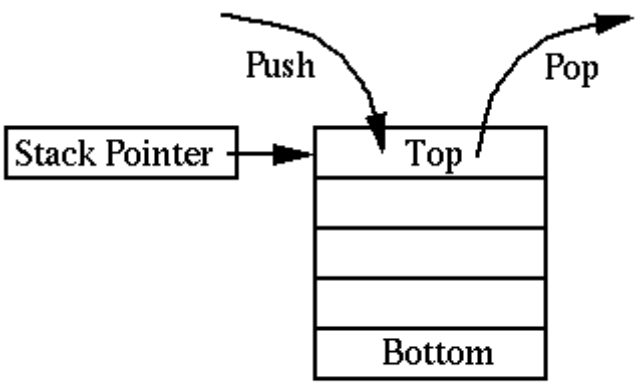
stack for the longest period of time. So, it can be simply seen to follow LIFO order.

- Mainly the following three basic operations are performed in the stack:
    1. **Push:** Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition.
    2. **Pop:** Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.
    3. **Peek or Top:** Returns top element of stack.
    4. **isEmpty:** Returns true if stack is empty, else false.

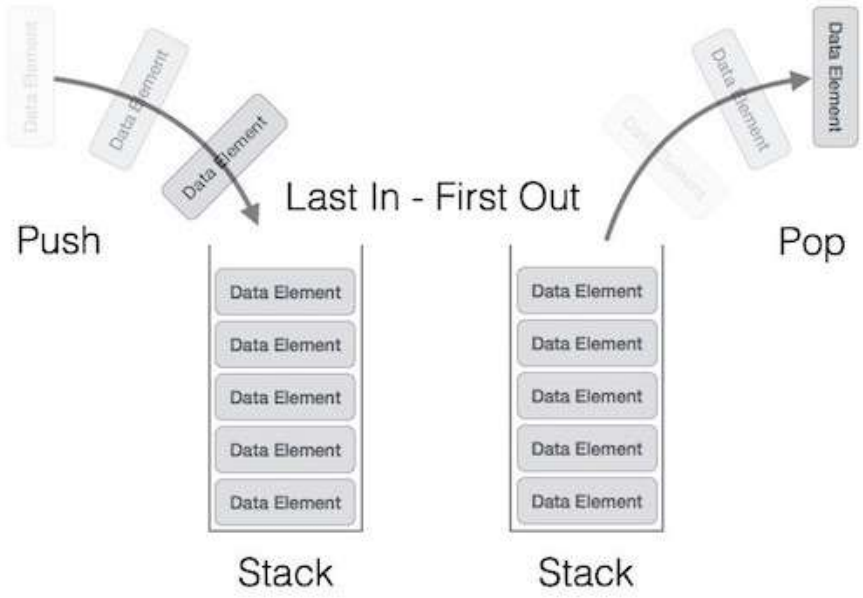
- Time Complexities of operations on stack:
  push(), pop(), isEmpty() and peek() all take O(1) time. We do not run any loop in any of these operations.

- **Applications of Stack:**
    1. Balancing of symbols
    2. Infix to Postfix /Prefix conversion
    3. Redo-undo features at many places like editors, photoshop.
    4. Forward and backward feature in web browsers.
    5. Used in many algorithms like Tower of Hanoi, tree traversals, stock span problem, histogram problem.
    6. Other applications can be Backtracking, Knight tour problem, rat in a maze, N queen problem and sudoku solver.
    7. In Graph Algorithms like Topological Sorting and Strongly Connected Components.

- **Implementation:**
  There are two ways to implement a stack:
    1. Using array
    2. Using linked list



LIFO Stack



Implementation Details:

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
char stack[50];
int top = -1;
void push(char x)
{
    stack[++top] = x;
}

char pop()
{
    if(top == -1)
        return -1;
    else
        return stack[top--];
}

int precedence(char x)
{
    if(x == '(')
        return 0;
    if(x == '+' || x == '-')
        return 1;
    if(x == '*' || x == '/')
        return 2;
    if(x== '^')
```

```
      return 3;
}
int evaluate(char *postfix)
{
  char ch;
  int i=0, operand1, operand2;
  while((ch=postfix[i++])!=0){
    if(isalnum(ch))
        push(ch-'0');
    else{
      operand2=pop();
      operand1=pop();
      switch(ch){
        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 stack[top];
}

void main()
{
    char exp[50],postfix[50];
    char *e, ch;
    int i=0;
    printf("Enter the expression : ");
    scanf("%s",exp);
    e = exp;
    printf("Postfix Expression: \n");
    while(*e != '\0')
    {
        if(isalnum(*e)){
            printf("%c ",*e);
            postfix[i++]=*e;
        }
```

```
        else if(*e == '(')
            push(*e);
        else if(*e == ')')
        {
            while((ch = pop()) != '('){
                printf("%c ", ch);
                postfix[i++]=ch;
            }
        }
        else
        {
            while(precedence(stack[top]) >= precedence(*e)){
                ch=pop();
                printf("%c ",ch);
                postfix[i++]=ch;
            }
            push(*e);
        }
        e++;
    }
    while(top != -1)
    {
        ch=pop();
        printf("%c", ch);
        postfix[i++]=ch;
    }
  printf("\nValue of expression: %d",evaluate(postfix));
}
```

For verification, my code is available on:
https://repl.it/@ARGHYADEEPDAS/DSExpt3

OUTPUT SCREEN:

```
Enter the expression :  (5*2+3-2)
Postfix Expression:
5 2 * 3 + 2 -
Value of expression: 11
```

CONCLUSION:

The program ran successfully as we were able to convert a given infix expression to postfix expression and then resolve the value of postfix expression.