Experiment No.3
Evaluate Postfix Expression using Stack ADT.
Name: Apurv Kini
Roll No:65
Date of Performance:
Date of Submission:
Marks:
Sign:

Experiment No. 3: Evaluation of Postfix Expression using stack ADT

Aim : Implementation of Evaluation of Postfix Expression using stack ADT

Objective:

1) Understand the use of Stack.
2) Understand importing an ADT in an application program.
3) Understand the instantiation of Stack ADT in an application program.
4) Understand how the member functions of an ADT are accessed in an application program

Theory:

An arithmetic expression consists of operands and operators. For a given expression in a postfix form, stack can be used to evaluate the expression. The rule is whenever an operand comes into the string, push it onto the stack and when an operator is found then the last two elements from the stack are popped and computed and the result is pushed back onto the stack. One by one the whole string of postfix expressions is parsed and the final result is obtained at an end of computation that remains in the stack.

Algorithm

```
Step 1: Add a ")" at the end of the postfix expression
Step 2: Scan every character of the postfix expression and repeat Steps 3 and 4 until ")"is
encountered
Step 3: IF an operand is encountered, push it on the stack
IF an operator 0 is encountered, then
        a. Pop the top two elements from the stack as A and B as A and B
        b. Evaluate BOA, where A is the topmost element and B is the element below A.
        c. Push the result of evaluation on the stack [END OF IF]
Step 4: SET RESULT equal to the topmost element of the stack
Step 5: EXIT
```

Code:

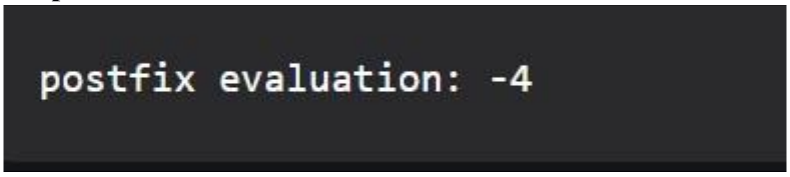
```
// C program to evaluate value of a postfix expression

#include <ctype.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>


// Stack type

struct Stack {

         int top;  unsigned

        capacity;  int*

        array;

};


// Stack Operations  struct Stack*

createStack(unsigned capacity)

{
```

```
            struct Stack* stack
                      =  (struct Stack*)malloc(sizeof(struct Stack));

            if (!stack)
                        return NULL;

            stack->top = -1;  stack-
           >capacity = capacity;  stack-
           >array
                       =  (int*)malloc(stack->capacity * sizeof(int));

            if (!stack->array)
                       return NULL;

            return stack;
}

int isEmpty(struct Stack* stack)
{
            return stack->top == -1;
}

char peek(struct Stack* stack)
{
            return stack->array[stack->top];
}
```

```
 char pop(struct Stack* stack)
{
            if (!isEmpty(stack))  return stack-
           >array[stack->top--];  return '$';
}

 void push(struct Stack* stack, char op)
{
            stack->array[++stack->top] = op;
 }

 // The main function that returns
value  // of a given postfix expression
int evaluatePostfix(char* exp)
 {
            // Create a stack of capacity equal to expression size
           struct Stack* stack = createStack(strlen(exp));
            int i;

            // See if stack was created
           successfully  if (!stack)  return -1;

            // Scan all characters one by one
            for (i = 0; exp[i]; ++i) {

                       // If the scanned character is an operand
```

```
                // (number here), push it to the stack.
                if (isdigit(exp[i]))  push(stack,
                        exp[i] - '0');

                // If the scanned character is an operator,  // pop
                two elements from stack apply the operator
                else {
                         int val1 = pop(stack);  int val2
                        = pop(stack);  switch (exp[i]) {
                        case '+':  push(stack, val2 +
                        val1);  break;  case '-':
                        push(stack, val2 - val1);
                        break;  case '*':  push(stack,
                        val2 * val1);  break;
                         case '/':  push(stack, val2 /
                                val1);  break;
                         }
                }
        }
        return pop(stack);
}

// Driver code
int main()
{
        char exp[] = "231*+9-";
        // Function call  printf("postfix evaluation: %d",
        evaluatePostfix(exp));  return 0;
}
```

Output:



```
postfix evaluation: -4
```

Conclusion:

Q.Elaborate the evaluation of the following postfix expression in your program.

AB+C-

Q.Will this input be accepted by your program. If so, what is the output?

To evaluate the postfix expression " AB+C-," follow these steps:

1. - When you encounter an operand (A), push it onto the stack.

2.- When you encounter another operand (B), push it onto the stack.

3 . When you encounter an operator (+), pop the top two operands from the stack (B and A in this case), perforrn the addition (A + B), and push the result back onto the stack.

4 . Continue scanning tkæ expression.

5 . When you encounter the next operator (-), pop tlE top two operands from the stack (result of A + B and C in this case), perform ffe subtraction (A + B - C), and push the final result &1Ck onto stack.

6. After processing entire expression, the final result is the only item left in the stack, which is "A + B - C."