MIT WORLD PEACE UNIVERSITY

Fundamental Data Structures
Second Year B. Tech, Semester 1

EXPRESSION EVALUATION USING STACK

PRACTICAL REPORT
ASSIGNMENT 8

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

November 28, 2022

Contents

1 Aim

Writing a C Program to Evaluate a Postfix Expression using Stack.

2 Objectives

1. To study Stack and its operations
2. To study the importance of expression evaluation

3 Problem Statements

Write a C Program to Evaluate a Postfix Expression using Stack.

4 Theory

Example of Working of Evaluation: 2 10 + 9 6 - /

1. First push 2 and 10 on the stack

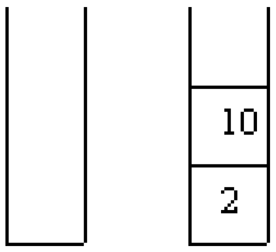

Figure 1: After 2 and 10 are pushed

2. Calculate 2 + 10 and then push 12 on the stack

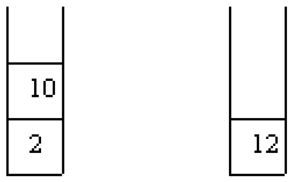

Figure 2: After 12 is pushed

3. Then push 6 and 9 on the stack

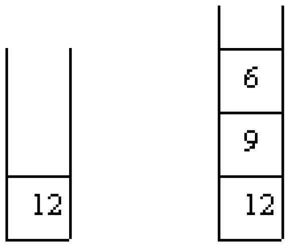

Figure 3: After 6 and 9 are pushed

4. Then Calculate 9 - 6 and push 3 on the stack

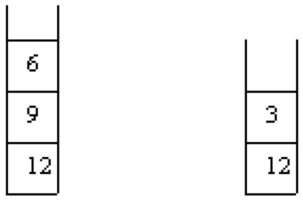

Figure 4: After 3 and 12 are pushed

5. Finally Calculate 12 / 3 and push 4 on the stack

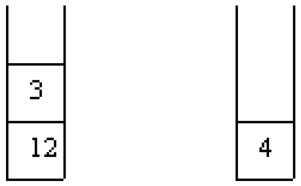

Figure 5: After result is pushed

5 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Visual Studio Code
Compilers : gcc on linux for C

6 Input

The Postfix Condition

7 Output

The Evaluation of the Postfix Expression entered.

8 Test Conditions

The Postfix Expression and Checking its Evaluation.

9 Code

9.1 Pseudo Code for calc

```
calc(op1, op2, op)
  ans;
  switch (op)
    case '+':
      ans = op1 + op2;
      break;
    case '-':
      ans = op1 - op2;
      break;
    case '*':
      ans = op1 * op2;
      break;
    case '/':
      ans = op1 / op2;
      break;
    case '%':
      ans = op1 % op2;
      break;
    case '^':
      ans = powerr(op1, op2);
      break;
    default:
      printf("\nInvalid operator");
      break;
  return ans;
```

9.1.1 Pseudo Code for Eval

```
eval(char post[MAX_SIZE])
  int z = 0, ans = 0, op1, op2;
  for (int i = 0; post[i] != 0; i++)
    if(ISALPHA(post[i]))
      print("\nEnter value of %c: ", post[i]);
      scan("%d", &z);
      push(z);
    else
      op1 = pop();
      op2 = pop();
      ans = calc(op2, op1, post[i]);
      push(ans);
  print("\nAnswer = %d\n", stack[top]);
```

9.2 C Implementation of Problem Statement

```
#include<stdio.h>

```

```c
#define MAX_SIZE 30
int stack[MAX_SIZE];
int top = -1;

int powerr(int a, int power){
    int i, result = 1;
    for(i = 0; i < power; i++){
        result *= a;
    }
    return result;
}

int isFull()
{
    if (top == MAX_SIZE - 1)
        return 1;
    else
        return 0;
}
int isEmpty()
{
    if (top == -1)
        return 1;
    else
        return 0;
}

int push(int item)
{
    if (!isFull())
    {
        top++;
        stack[top] = item;
    }
    else
    {
        printf("\nSTACK OVERFLOW!\n");
    }
}
int pop()
{
    if (isEmpty())
    {
        printf("Stack is Empty \n\n STACK UNDERFLOW!!");
        return 0;
    }
    else
    {
        // printf("Removed this thing %c\n", stack[top]);
        top--;
        return stack[top + 1];
    }
}


int calc(int op1, int op2, char op)
{
    int ans;
    switch (op)
    {
    case '+':
        ans = op1 + op2;
        break;
    case '-':
        ans = op1 - op2;
        break;
    case '*':
        ans = op1 * op2;
        break;
    case '/':
        ans = op1 / op2;
        break;
    case '%':
        ans = op1 % op2;
        break;
    case '^':
        ans = powerr(op1, op2);
        break;
    default:
        printf("\nInvalid operator");
        break;
    }
    return ans;
}

void eval(char post[MAX_SIZE])
{
    int z = 0, ans = 0, op1, op2;
    for (int i = 0; post[i] != 0; i++)
    {
        if(post[i] >= 97 && post[i] <= 122)
        {
            printf("\nEnter value of %c: ", post[i]);
            scanf("%d", &z);
            push(z);
        }
        else
        {
            op1 = pop();
            op2 = pop();
            ans = calc(op2, op1, post[i]);
            push(ans);
        }
    }
    printf("\nAnswer = %d\n", stack[top]);
}

int main()
{
    char post[30];
    printf("Enter the postfix expression: ");
    scanf("%s", post);
    eval(post);
}
```

Listing 1: Main.Cpp

9.3 Input and Output

```
Enter the postfix expression: abc+-cdb/+*b^c+

Enter value of a: 6

Enter value of b: 2

Enter value of c: 3

Invalid operator
Enter value of c: 3

Enter value of d: 8

Enter value of b: 2

Enter value of b: 2

Enter value of c: 3

Answer = 52
```

Listing 2: Output

10 Time Complexity

- clac() : O(n)
- eval() : O(n) where n is the length of the post array.

11 Conclusion

Thus, implemented postfix expression evaluation using stack data structure.

12 FAQs

1. **How does prefix expression evaluation work?**
   In this notation, operator is prefixed to operands, i.e. operator is written ahead of operands. For example, +ab. This is equivalent to its infix notation a + b. Prefix notation is also known as Polish Notation.

   (a) Step 1: Start from the last element of the expression.
   (b) Step 2: check the current element.
   (c) Step 2.1: if it is an operand, push it to the stack.
   (d) Step 2.2: If it is an operator, pop two operands from the stack. Perform the operation and push the elements back to the stack.
   (e) Step 3: Do this till all the elements of the expression are traversed and return the top of stack which will be the result of the operation.

2. **What is the advantage of prefix and postfix expressions?**

(a) Prefix notations are needed when we require operators before the operands while postfix notations are needed when we require operators after the operands.

(b) Prefix notations are used in many programming languages like LISP.

(c) Prefix notations and Prefix notations can be evaluated faster than the infix notation.

(d) Postfix notations can be used in intermediate code generation in compiler design.

(e) Prefix and Postfix notations are easier to parse for a machine.

(f) With prefix and postfix notation there is never any question like operator precedence.

(g) There is no issue of left-right associativity.