Harsh Kathuria
2003055
C21

DATE 22/09/21 PAGE NO.

Experiment 3

Aim: Evaluate Postfix Expression using Stack ADT.

Theory:

Algorithm:

```
Evaluate Postfix(E)
{
1. Create an Empty stack(S)
2. for i=0 to len(E)-1
   {
3. Case 1:
     if (E[i] is operand)    // if operand is encountered, push it onto
4.  {push(E[i])}                 stack.

5. case 2:
     Else if (E[i] is operator)  // if operator is encountered, pop2
   {                                elements.
6.   op2 = pop()      // Top element
7.   op1 = pop()      // Next to top element.
8.   res = perform(op1, op2, E[i])   // result = op1 operator op2
9.   push(res)
     }
   }  // End of for.
10. return top of stack.
  }
```

Harsh Kashiwal
2003035

Example. 6821 +-684 /+*

Token	Action	Stack
-	-	Empty.
6	push	6
2	push	6,2
1	push	6,2,1
+	Pop 1,2 ; 1+2=3	6,3
	push (3)	
-	Pop 3,6 ; 6-3=3	3
	push (3)	
6	push	3,6
8	push	3,6,8
4	push	3,6,8,4
/	Pop 4,8 ; 8/4=2	3,6,2
	push (2)	
+	Pop 2,6 ; 6+2=8	3,8
	push (8)	
*	Pop 8,3 ; 3*8=24	24
	push (24)	

Result = 24

Conclusion :- The expressions written in postfix form are evaluated faster compared to infix notation as parenthesis are not required in postfix.

PROGRAM: Write a program to implement postfix evaluation using stack

```
#include <iostream>
#include <string.h>
using namespace std;
#define SIZE 100

int stack[SIZE];
int Top = -1;
void push(char Item)
{
    if (Top >= SIZE - 1)
    {
        cout << "\nStack Overflow.";
    }
    else
    {
        Top = Top + 1;
        stack[Top] = Item;
    }
}

void push1(int y)
{
    stack[++Top] = y;
}

void get_stack()
{
    int i;
    for (i = 0; i <= Top; i++)
    {
        cout << stack[i];
        cout << " ";
    }
}

char pop()
{
    char Item;
    if (Top < 0)
    {
        cout << "Stack Under Flow: Invalid Infix Expression";
        getchar();
        exit(1);
    }
    else
    {
        Item = stack[Top];
```

```
        Top = Top - 1;
        return (Item);
    }
}
int pop1()
{
    return stack[Top--];
}
int is_operator(char symbol)
{
    if (symbol == '^' || symbol == '*' || symbol == '/' || symbol == '+' || sy
mbol == '-')
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

int precedence(char symbol)
{
    if (symbol == '^')
    {
        return (3);
    }
    else if (symbol == '*' || symbol == '/')
    {
        return (2);
    }
    else if (symbol == '+' || symbol == '-')
    {
        return (1);
    }
    else
    {
        return (0);
    }
}

void InfixToPostfix(char infix_exp[], char postfix_exp[])
{
    int i, k;
    char Item;
    char x;

    push('(');
```

```
    strcat(infix_exp, ")");

    i = 0;
    k = 0;
    Item = infix_exp[i];
    while (Item != '\0')
    {
        if (Item == '(')
        {
            push(Item);
        }
        else if (isalnum(Item))
        {
            postfix_exp[k] = Item;
            k++;
        }
        else if (is_operator(Item) == 1)
        {
            x = pop();
            while (is_operator(x) == 1 && precedence(x) >= precedence(Item))
            {
                postfix_exp[k] = x;
                k++;
                x = pop();
            }
            push(x);

            push(Item);
        }
        else if (Item == ')')
        {
            x = pop();
            while (x != '(')
            {
                postfix_exp[k] = x;
                k++;
                x = pop();
            }
        }
        else
        {
            cout << "\nInvalid Infix Expression.\n";
            getchar();
            exit(1);
        }
        i++;

        Item = infix_exp[i];
```

```
    }
    if (Top > 0)
    {
        cout << "\nInvalid Infix Expression.\n";
        getchar();
        exit(1);
    }
    postfix_exp[k] = '\0';
}

int main()
{
    int i, ch;
    char exp[SIZE];
    char *a;
    int n1, n2, n3, num;
    char infix[SIZE], postfix[SIZE];
    cout << "You can enter infix or postfix expression, choose an option\n";
    cout << "1. Infix expression\n2. Postfix Expression\n\nEnter an Option:
 ";
    cin >> ch;
    switch (ch)
    {
    case 1:
        for (i = 0; i < SIZE; i++)
        {
            postfix[i] = '\0';
        }
        cout << "\nYou have chosen 1, Enter an infix expression:    ";
        cin >> infix;
        InfixToPostfix(infix, postfix);
        cout << "\n";
        cout << "Resultant postfix expression:    ";
        puts(postfix);
        a = postfix;
        break;
    case 2:
        cout << "Enter Postfix Expression : " << endl;
        cin >> postfix;
        a = postfix;
        break;
    }

    cout << "\nToken\tStack\n";
    char token;
    while (*a != '\0')
    {
        if (isdigit(*a))
```

```
        {
            num = *a - '0';
            token = *a;
            push1(num);
        }
        else
        {
            n1 = pop1();
            n2 = pop1();
            switch (*a)
            {
            case '+':
            {
                n3 = n1 + n2;
                token = '+';
                break;
            }
            case '-':
            {
                n3 = n2 - n1;
                token = '-';
                break;
            }
            case '*':
            {
                n3 = n1 * n2;
                token = '*';
                break;
            }
            case '/':
            {
                n3 = n2 / n1;
                token = '/';
                break;
            }
            }
            push1(n3);
        }
        cout << " \n " << token << "\t";
        get_stack();
        cout << " \n ";

        a++;
    }
    cout << "Final Result: " << pop1();
    return 0;
}
```

OUTPUT:

```
PS C:\Users\Harsh\OneDrive\Desktop\DS\CODES> cd "c:\Users\Harsh\OneDrive\Desktop\DS\CODES\"
 { .\Evaluation }
You can enter infix or postfix expression, choose an option
1. Infix expression
2. Postfix Expression

Enter an Option:    1

You have chosen 1, Enter an infix expression:      (5+4)*2

Resultant postfix expression:    54+2*

Token   Stack

 5      5

 4      5 4

 +      9

 2      9 2

 *      18
 Final Result: 18
PS C:\Users\Harsh\OneDrive\Desktop\DS\CODES>
```