Experiment 1
Date 20-10-2022

INFIX TO POSTFIX CONVERSION WITH STACK

Aim

Write a C program to convert an infix expression to a postfix expression and evaluate the postfix expression using stack.

1 Infix to Postfix Algorithm

1.1 Algorithm

```
Infix_To_Postfix()
Step 1 : Scan the Infix Expression from left to right.
Step 2 : If the scanned character is an operand,
        append it with final Infix to Postfix string.
Step 3 : Else,
Step 3.1 :  If the precedence order of the scanned(incoming) operator is
            greater than the precedence order of the operator in the stack
          (or the stack is empty or the stack contains a ‘(‘ , push it on stack.
Step 3.2 : Else, Pop all the operators from the stack which are greater than
            or equal to in precedence than that of the scanned operator.
            After doing that Push the scanned operator to the stack.
            (If you encounter parenthesis while popping then stop there and
            push the scanned operator in the stack.)
Step 4 : If the scanned character is an ‘(‘ , push it to the stack.
Step 5 : If the scanned character is an ‘)’, pop the stack and and output it until a ‘(‘ res
Step 6 : Repeat steps 2-6 until infix expression is scanned.
Step 7 : Print the output
Step 8 : Pop and output from the stack until it is not empty.

Evaluation()
Step 1:Create a stack to store operands.
Step 2:      Scan the given expression from left to right.
Step 3:      a)If the scanned character is an operand, push it into the stack.
             b)If the scanned character is an operator, POP 2 operands from stack
             and perform operation and PUSH the result back to the stack.
```

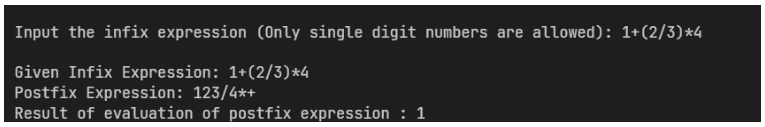
```
step 4:      Repeat step 3 till all the characters are scanned.
step 5:      When the expression is ended, the number in the stack is the final result.
```

1.2 Program

```
#include <stdio.h>
#include <math.h>
char postfix[50], infix[50];
int top = -1, stack[50], flag = 1;
void push(int elem)
{
    stack[++top] = elem;
}
int pop()
{
    return (stack[top--]);
}
int pr(char elem)
{
    switch (elem)
    {
    case '(':
        return 0;
    case '+':
    case '-':
        return 1;
    case '*':
    case '/':
        return 2;
    case '^':
        return 3;
    }
}
void infix_to_postfix()
{
    char ch, ele;
    int i = 0, k = 0;
    ch = infix[i++];
    push('(');
    while (ch != '\0')
    {
        if (ch == '(')
            push(ch);
```

```
         else if (ch >= '0' && ch <= '9')
            postfix[k++] = ch;
        else if ((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z'))
        {
            postfix[k++] = ch;
            flag = 0;
        }
        else if (ch == ')')
        {
            while (stack[top] != '(')
                postfix[k++] = pop();
            ele = pop();
        }
        else
        {
            if (ch != '^')
            {
                while (pr(stack[top]) >= pr(ch) && top != -1)
                    postfix[k++] = pop();
                push(ch);
            }
            else
                push(ch);
        }


            ch = infix[i++];
    }
    while (stack[top] != '(')
        postfix[k++] = pop();
    postfix[k] = '\0';
}

int eval_postfix()
{
    char ch;
    int i = 0, op1, op2, t;
    ch = postfix[i++];
    while (ch != 0)
    {
        if (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '^')
        {
            op2 = pop() - 0;
            op1 = pop() - 0;
            switch (ch)
            {
            case '+':
                t = op1 + op2;
                break;
            case '-':
                t = op1 - op2;
                break;
            case '*':
                t = op1 * op2;
                break;
            case '/':
                t = op1 / op2;
                break;
            case '^':
                t = pow(op1, op2);
                break;
            }
            push(t);
        }

        else
        {
            ch = ch - '0';
            push(ch);
        }
        ch = postfix[i++];
    }
    return stack[top];
}

int main()
{
    printf("\nInput the infix expression (Only single digit numbers are allowed): ");
    scanf("%s", infix);
    infix_to_postfix();
    printf("\nGiven Infix Expression: %s \nPostfix Expression: %s", infix, postfix);
    top = -1;
    if (flag)
        printf("\nResult of evaluation of postfix expression : %d", eval_postfix(postfix));
}
```

1.3 Sample Input and Output



```
Input the infix expression (Only single digit numbers are allowed): 1+(2/3)*4

Given Infix Expression: 1+(2/3)*4
Postfix Expression: 123/4*+
Result of evaluation of postfix expression : 1
```

Figure 1: Output

1.4 Result

Successfully executrd progarm to convert Infix expression to postfix expression and to evaluated .