Experiment No 2: Conversion of Infix to postfix expression using stack ADT

Aim: To convert infix expression to postfix expression using stack ADT

Objective:

1) Understand the use of stack
2) Understand how to import an ADT in an application program
3) Understand the instantiation of stack ADT in an application program
4) Understand how the member function of an ADT are accessed in an application program

Theory: To convert infix expression to postfix expression, use the stack data structure. Scan the infix expression from left to right. Whenever we get an operand, add it to the postfix expression and if we get an operator or parenthesis add it to the stack by maintaining their precedence.

Infix

Infix Expression contains operator in-between every pair of operands, Expression of the form a op b.

Postfix

Postfix Expression contains operator followed for every pair of operands, Expression of the form a b op.

Algorithm:

- **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 '(' or '[' or '{'), 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 '(' or '[' or '{', push it to the stack.
- **Step 5** : If the scanned character is an ')'or ']' or '}', pop the stack and and output it until a '(' or '[' or '{' respectively is encountered, and discard both the parenthesis.
- **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.

Code :-

```c
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#define SIZE 100
char stack[SIZE] ;
int top = -1 ;
void push (char item) {
 if (top >= SIZE-1)
   printf ("\nStack verification") ;
 else {
   top++;
   stack [top] = item ;
 }
}
char pop () {
 char item ;
 if (top < 0) {
   printf ("Stack is underflow") ;
   getchar () ;
   exit (1) ;
 }
 else {
   item = stack[top] ;
   top-- ;
```

```c
    return (item) ;
  }
}

int is_operator (char symbol) {
  if (symbol == '+' || symbol == '^' || symbol == '*' || symbol == '/' || symbol == '-')
    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, j ;
  char item ;
  char x ;
  push ('(') ;
  strcat (infix_exp, ")") ;
  i = 0 ;
  j = 0 ;
  item = infix_exp[i] ;
  while (item != '\0') {
    if (item == '(') {
      push (item) ;
    }
    else if (isdigit (item) || isalpha (item)) {
      postfix_exp[j] = item ;
      j++ ;
    }
    else if (is_operator(item) == 1) {
      x = pop() ;
      while (is_operator(x) == 1 && precedence(x) >= precedence(item)) {
          postfix_exp[j] = x ;
          j++ ;
          x = pop() ;
      }
      push (x) ;
      push (item) ;
```

```c
    }
    else if (item == ')') {
     x = pop() ;
     while (x != '(') {
          postfix_exp[j] = x ;
          j++ ;
          x = pop() ;
     }
    }
    else {
     printf ("\nInvalid infix Expression\n") ;
     getchar () ;
     exit (1) ;
    }
    i++ ;
    item = infix_exp[i] ;
  }
  if (top>0) {
    printf ("\nInvalid infix Expression\n") ;
    getchar () ;
    exit (1) ;
  }
  postfix_exp[j] = '\0' ;
}

int main () {
  char infix [SIZE], postfix [SIZE] ;
  printf ("ASSUMPTION: The infix expression contains single letter variables and single digit constants
only.\n") ;
  printf ("\nEnter Infix Expression: ") ;
  gets (infix) ;

  infixtopostfix (infix, postfix) ;
  printf ("Postfix Expression: ") ;
  puts (postfix) ;
  return 0 ;
}
```

Output

```
ASSUMPTION: The infix expression contains single letter variables and single dig
it constants only.

Enter Infix Expression: a+b-c*d/e
Postfix Expression: ab+cd*e/-
_
```

Conclusion

We have learned about the infix to postfix conversion using stack in C. We have discussed the algorithm with the dry to convert the infix to postfix using stack in C. Hope this blog helps you understand and solve the problem