**TITLE 36**

Write a C program to convert the given infix expression to post-fix expression using STACK

**OBJECTIVE:**

By the end of this problem we will be able to convert a given infix expression to post-fix expression using STACK

**PROBLEM STATEMENT:**

In this problem with the help of STACK we will convert an infix expression to post-fix expression.

The output is printed if the program written doesn’t have any kind of errors.

**ALGORITHM:**

START

Define variables: top

INPUT: Read from the keyboard

COMPUTATION: Computing the infix to post-fix expression

DISPLAY: Displaying the converted post-fix expression

STOP

**PROGRAM:**

#include <stdio.h>  
#include <string.h>  
#include <stdlib.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));  
  
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;  
}  
  
  
// A utility function to check if the given character is operand  
int isOperand(char ch)  
{  
return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z');  
}  
  
// A utility function to return precedence of a given operator  
// Higher returned value means higher precedence  
int Prec(char ch)  
{  
switch (ch)  
{  
case '+':  
case '-':  
return 1;  
  
case '\*':  
case '/':  
return 2;  
  
case '^':  
return 3;  
}  
return -1;  
}  
  
  
// The main function that converts given infix expression  
// to postfix expression.  
int infixToPostfix(char\* exp)  
{  
int i, k;  
  
// Create a stack of capacity equal to expression size  
struct Stack\* stack = createStack(strlen(exp));  
if(!stack) // See if stack was created successfully  
return -1 ;  
  
for (i = 0, k = -1; exp[i]; ++i)  
{  
// If the scanned character is an operand, add it to output.  
if (isOperand(exp[i]))  
exp[++k] = exp[i];  
  
// If the scanned character is an ‘(‘, push it to the stack.  
else if (exp[i] == '(')  
push(stack, exp[i]);  
  
// If the scanned character is an ‘)’, pop and output from the stack  
// until an ‘(‘ is encountered.  
else if (exp[i] == ')')  
{  
while (!isEmpty(stack) && peek(stack) != '(')  
exp[++k] = pop(stack);  
if (!isEmpty(stack) && peek(stack) != '(')  
return -1; // invalid expression  
else  
pop(stack);  
}  
else // an operator is encountered  
{  
while (!isEmpty(stack) && Prec(exp[i]) <= Prec(peek(stack)))  
exp[++k] = pop(stack);  
push(stack, exp[i]);  
}  
  
}  
  
// pop all the operators from the stack  
while (!isEmpty(stack))  
exp[++k] = pop(stack );  
  
exp[++k] = '\0';  
printf( "%s", exp );  
}  
  
// Driver program to test above functions  
int main()  
{  
char exp[] = "a+b\*(c^d-e)^(f+g\*h)-i";  
infixToPostfix(exp);  
return 0;  
}

**CONCLUSION:**

The simulation of the above C program helps us understand how we can convert an infix expression to post-fix expression with the usage of STACK.

**OUTPUT:**

abcd^e-fgh\*+^\*+i-