EXPERIMENT NO. 2

CONVERSION OF AN EXPRESSION FROM INFIX TO POSTFIX

Aim: Write A Program To Convert An Infix Expression To Postfix Expression.

Theory: Infix, Postfix & Prefix Are 3 Different But Equivalent Notations Of Writing Algebraic Expressions. Postfix Notation Was Developed By **Jan Lukasiewicz** Who Was A **Polish Logician, Mathematician & Philosopher**. His Aim Was To Develop A Parenthesis – Free Prefix Notation & A Postfix Notation, Which Is Better Known As Reverse Polish Notation Or RPN.

DEFINITIONS:

Infix expression: The Expression of the form a op b. When an operator is inbetween every pair of operands.

Postfix expression: The Expression of the form a b op. When an operator is followed for every pair of operands.

Why postfix representation of the expression?

The compiler scans the expression either from left to right or from right to left. Consider the below expression: a op1 b op2 c op3 d

If op1 = +, op2 = *, op3 = +

The compiler first scans the expression to evaluate the expression b * c, then again scan the expression to add a to it. The result is then added to d after another scan.

The repeated scanning makes it **very in-efficient**. It is better to convert the expression to postfix (or prefix) form before evaluation.

The corresponding expression in postfix form is: abc*d++.The postfix expressions can be evaluated easily using a stack.

EXAMPLE:

Infix to Postfix Conversion

- 3+4*5/6
- Stack: - • Output:• 3+4*5/6
- Stack: • Output: 3
- 3+4*5/6
- Stack: + • Output: 3
- 3+4*5/6
- Stack: + • Output: 3 4
- 3+4*5/6
- Stack: + * • Output: 3 4
- 3+4*5/6
- Stack: + * • Output: 3 4 5
- 3+4*5/6
- Stack: + • Output: 3 4 5 *
- 3+4*5/6
- Stack: + / • Output: 3 4 5 *
- 3+4*5/6
- Stack: + / • Output: 3 4 5 * 6
- 3+4*5/6
- Stack: + • Output: 3 4 5 * 6 /
- 3+4*5/6
- Stack: - • Output: 3 4 5 * 6 / +

Algorithm:

1) Start
2) We use a stack.
3) When an operand is read , output it
4) When an operator is read
   a) Pop until the top of the stack has an element of lower precedence
   b) Then push it
5) When) is found, pop until we find the matching (

6) (has the lowest precedence when in the stack
7) but has the highest precedence when in the input
8) When we reach the end of input, pop until the stack is empty
9) Exit.

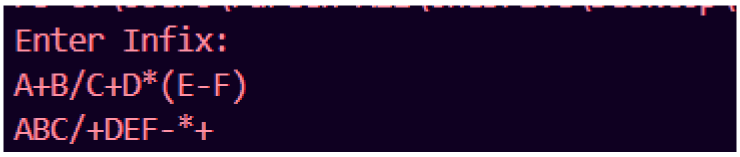
Program:

```
 #include<stdio.h>
#include<string.h>
int isoperand(char n)
{
if((n>='a'&&n<='z')||(n>='A'&&n<='Z') || (n>=0&&n<=9))
return 1;
else
return 0;
}

int priority(char n)
{ if(n=='*' || n=='/')
return 2;
else if(n=='+' || n=='-')
return 1;
else
return 0;
}
int main()
{ int j=-1,i,top=-1;
char infix[20],postfix[20],stack[20];
printf("Enter Infix:\n");
gets(infix);
for(i=0;infix[i]!='\0';i++)
{ if(infix[i]=='(')
{ top++;
stack[top]=infix[i];
}
else if(isoperand(infix[i])==1)
{ j=j+1;
postfix[j]=infix[i];
}
else if(top==-1)
{
top=top+1;
stack[top]=infix[i];
}
else if(infix[i]==')')
{ while(stack[top]!='(')
{ j=j+1;

postfix[j]=stack[top];
top=top-1;
}
top=top-1;
}
else if(priority(infix[i])>priority(stack[top]))
{ top=top+1;
stack[top]=infix[i];
}
else if(priority(infix[i])<priority(stack[top]))
{ while(priority(stack[top])>=priority(infix[i]) && top!=-1 && stack[top]!='(')
{ j=j+1;
postfix[j]=stack[top];
top=top-1;
}
top=top+1;
stack[top]=infix[i];
}
}
while(top!=-1)
{ j=j+1;
postfix[j]=stack[top];
top=top-1;
}
postfix[j+1]='\0';
puts(postfix);
return 0;
}
```

OutPut:



```
Enter Infix:
A+B/C+D*(E-F)
ABC/+DEF-*+
```