Experiment No.2
Convert an Infix expression to Postfix expression using stack ADT.
Name: Sharvari Anand Bhondekar
Roll No: 06
Date of Performance:
Date of Submission:
Marks:
Sign:

.

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 functions of an ADT are accessed in an application program.

Theory:

Postfix notation is a way of representing algebraic expressions without parentheses or operator precedence rules. In this notation, expressions are evaluated by scanning them from left to right and using a stack to perform the calculations. When an operand is encountered, it is pushed onto the stack, and when an operator is encountered, the last two operands from the stack are popped and used in the operation, with the result then pushed back onto the stack. This process continues until the entire postfix expression is parsed, and the result remains in the stack.

Conversion of infix to postfix expression

Expression	Stack	Output
2	Empty	2
*	*	2
3	*	23
/	/	23*
(	/(	23*
2	/(	23*2
-	/(-	23*2
1	/(-	23*21
)	/	23*21-
+	+	23*21-/
5	+	23*21-/5
*	+*	23*21-/53
3	+*	23*21-/53
	Empty	23*21-/53*+

Algorithm:

Conversion of infix to postfix

```
Step 1: Add ")" to the end of the infix expression
Step 2: Push "(" on to the stack
Step 3: Repeat until each character in the infix notation is scanned
         IF a "(" is encountered,push it on the stack
         IF an operand (whether a digit or a character) is encountered, add it to the
           postfix expression.
         IF a ")" is encountered, then
            a. Repeatedly pop from stack and add it to the postfix expression until a
           "(" is encountered.
           b.Discard the "(". That is, remove the "(" from stack and do not
           add it to the postfix expression
        IF an operator 0 is encountered, then
         a. Repeatedly pop from stack and add each operator (popped from the stack) to t postfix
            expression which has the same precedence or a higher precedence than o
         b. Push the operator o to the stack
         [END OF IF]
Step 4: Repeatedly pop from the stack and add it to the postfix expression until the stack is
empty
Step 5: EXIT
```

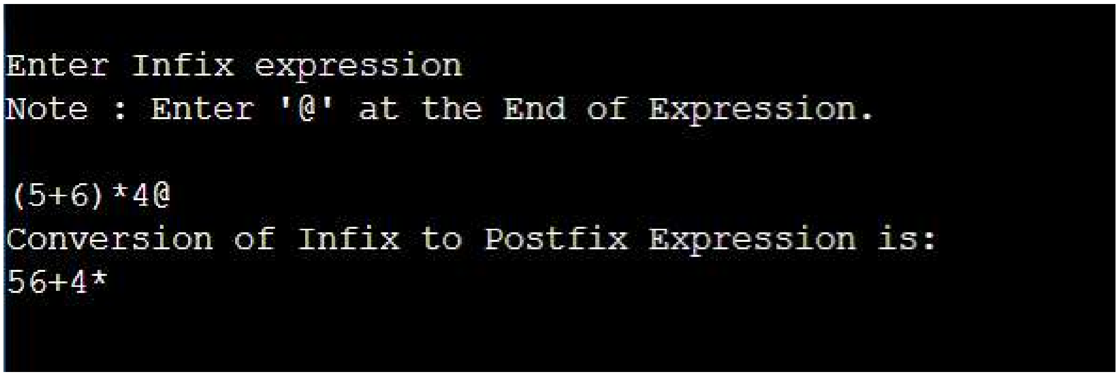
Code:

```
 #include <stdio.h>

#include <ctype.h>

#define MAX 100




int stack[MAX];

int Top = -1;


void push(char item)

{

if (Top >= MAX - 1)
{
printf("stack Overflow");
}
else
{
Top = Top + 1;
stack[Top] = item;
}
}


int pop()
{
int item;
if (Top ==-1)
{
printf("stack Underflow");
return 0;
}
else
{
item = stack[Top];
Top = Top - 1;
return item;
}
}
```

```
int priority(char x)
{
if(x == '(')
{
return 0;
}
if(x == '+' || x == '-')
{
return 1;
}
if (x == '^')
{
return 2;
}
if(x== '*' || x=='/' || x=='%')
{
return 3;
}
}
void ConPostfix(char postfix[])
{
int i;
char ch;
char x;
printf("Conversion of Infix to Postfix Expression is:\n");
for (i = 0; postfix[i] != '@'; i++)
```

```
{
ch = postfix[i];
if (isalnum(ch))
{
printf("%c",ch);
}
else if (ch=='(')
{
push(ch);
}
else if(ch==')')
{
while((x=pop()) != '(')
{
printf("%c",x);
}
}
else
{
while(priority(stack[Top]) >= priority(ch))
{
printf("%c",pop());
}
push(ch);
}
}
while(Top!=-1)
{
printf("%c",pop());
}
printf("\n");
}
void main()
{
int i;
char postfix[100];
printf(" \nEnter Infix expression\n");
printf("Note : Enter '@' at the End of Expression.\n\n");
for (i = 0; i <=99; i++)
{
scanf("%c", &postfix[i]);
if (postfix[i] == '@')
{
break;
}
}
ConPostfix(postfix);
}
```

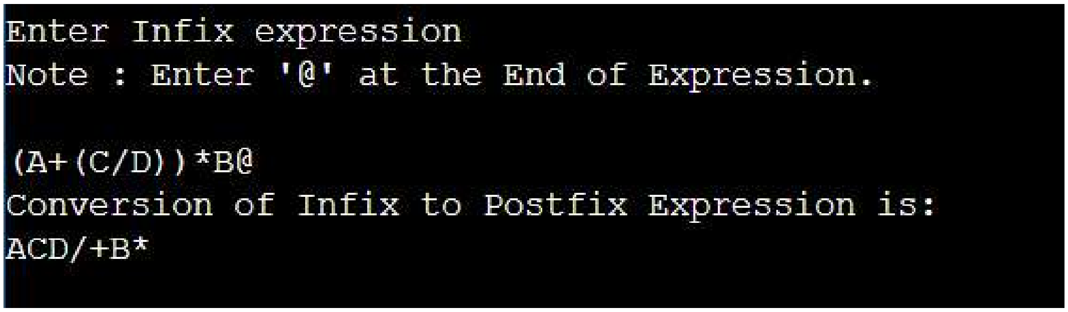
Output:



```
Enter Infix expression
Note : Enter '@' at the End of Expression.

(5+6)*4@
Conversion of Infix to Postfix Expression is:
56+4*
```

Conclusion:

1.)Convert the following infix expression to postfix (A+(C/D))*B



```
Enter Infix expression
Note : Enter '@' at the End of Expression.

(A+(C/D))*B@
Conversion of Infix to Postfix Expression is:
ACD/+B*
```

2.)How many push and pop operations were required for the above conversion?

➔ **5 push operations** and **5 pop operations** were required to convert the infix expression to postfix.

3.)Where is the infix to postfix conversion used or applied?

Infix to postfix conversion is primarily used in:

1. **Compilers and Interpreters**: For efficient expression evaluation during program compilation.
2. **Stack-Based Calculators**: To simplify calculation by avoiding parentheses and operator precedence.
3. **Expression Parsing**: In software to evaluate mathematical or logical expressions.
4. **Digital Circuit Design**: To implement arithmetic operations efficiently in hardware.

It simplifies the process of evaluating complex expressions by converting them into a stack-based, parenthesis-free format.