Experiment No.2
Aim: To convert infix expression to postfix expression using stack ADT.
Name: Nikhil Kamalaji Shingade
Roll no: 57
Date of Performance:
Date of Submission:

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>
#define MAX 100
int stack[MAX];
int top = -1;
void push(char item)
{
if (top >= MAX - 1) {
printf("stack over flow");
return;
}
else {
top = top + 1;
stack[top] = item;
  }
}
int pop()
{
int item;
if (top ==-1) {
printf("stack under flow");
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=='*' || 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());
  }
}
void main()
{
int i;
char postfix[100];
printf(" \nEnter Infix expression\n");
for (i = 0; i <=99; i++) {
 scanf("%c", &postfix[i]);
if (postfix[i] == '@')
 {
break;
    }
}
ConPostfix(postfix);
}
```

Output:

```
Output
/tmp/p46iUBezi3.o

Enter Infix expression
1+2/4@
Conversion of Infix to Postfix Expression is:
124/+

=== Code Exited With Errors ===
```

Conclusion:

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

Expression	Stack	Output
(	(	
A	(	A
+	(+	A
(	(+(	A
C	(+(	AC
/	(+(/	AC
D	(+(/	ACD
)	(+	ACD/
)		ACD/+
*	*	ACD/+
B	*	ACD/+B
		ACD/+B*

2) How many push and pop operations were required for the above conversion?
Ans:
To convert the infix expression (A+(C/D))*B to postfix A C D / + B *, the process of pushing and poppi ng operators involves the following operations:

1. Push ((1 push)
2. Push + (2 pushes)
3. Push ((3 pushes)
4. Push / (4 pushes)
5. Pop / (1 pop)
6. Pop + (2 pops)
7. Pop ((3 pops)
8. Push * (4 pushes)
9. Pop * (4 pops)

In total, 4 push operations and 4 pop operations to convert the given infix expression to postfix.

3) Where is the infix to postfix conversion used or applied?
Ans:
Converting infix to postfix is often used in calculators and programming to make expression evaluation easier. By converting to postfix (or Reverse Polish Notation), we remove the need for pare ntheses and follow a straightforward left-to
right evaluation. It simplifies the process, making things more efficient and less error-prone