Experiment No.2
Convert an Infix expression to Postfix expression using stack ADT.
Name:Apurv Kini
Roll No:65
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:

```
// C code to convert infix to postfix expression


#include <stdio.h>

#include <stdlib.h>

#include <string.h>


#define MAX_EXPR_SIZE 100


// Function to return precedence of operators

int precedence(char operator)

{

        switch (operator) {

        case '+':

        case '-':  return

                1;

        case '*':

        case '/':  return

                2;

        case '^':

        return 3;

        default:

                return -1;

        }
```

```
}

 // Function to check if the scanned character
 // is an operator  int
isOperator(char ch)
{
        return (ch == '+' || ch == '-' || ch == '*' || ch == '/'
                        || ch == '^');
}

 // Main functio to convert infix expression
 // to postfix expression  char*
infixToPostfix(char* infix)
 {
        int i, j;
        int len = strlen(infix);  char* postfix =
       (char*)malloc(sizeof(char) * (len + 2));  char
       stack[MAX_EXPR_SIZE];
        int top = -1;

        for (i = 0, j = 0; i < len; i++) {  if
                (infix[i] == ' ' || infix[i] == '\t')
                continue;

                 // If the scanned character is operand
                 // add it to the postfix expression
```

```
 if (isalnum(infix[i])) {
            postfix[j++] = infix[i];
}

 // if the scanned character is
'('  // push it in the stack  else
if (infix[i] == '(') {
stack[++top] = infix[i];
}

 // if the scanned character is ')'
 // pop the stack and add it to the  // output
string until empty or '(' found  else if
(infix[i] == ')') {  while (top > -1 &&
stack[top] != '(')  postfix[j++] = stack[top--
];

                    top--;
 }

 // If the scanned character is an operator
 // push it in the stack  else if
(isOperator(infix[i])) {
while (top > -1
                    && precedence(stack[top])
                                    >= precedence(infix[i]))
```

```
                    postfix[j++] = stack[top--];
                stack[++top] = infix[i];
            }
        }

        // Pop all remaining elements from the
        stack  while (top > -1) {  if (stack[top] == '('
        {  return "Invalid Expression";
            }
            postfix[j++] = stack[top--];
        }
        postfix[j] = '\0';
        return postfix;
}

// Driver code
int main()
{
        char infix[MAX_EXPR_SIZE] = "a+b*(c^d-e)^(f+g*h)-i";

        // Function call  char* postfix =
        infixToPostfix(infix);
        printf("%s\n", postfix);
        free(postfix);  return 0;}
```

Output:

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

Conclusion:

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

scanned	stack	pe
(	(	empty
A	(	A
+	+	A
(	+(	A
C	+(	AC
/	+(/	AC
D	+(/	ACD
)	+	ACD/
)		ACD/+
*	*	ACD/+
B	*	ACD/+B
		ACD/+B*

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

In the given conversion of the infix expression "A+(CD)*B" to postfix notation
Push Operations: 9
Pop Operations: 9

There were a total of9 push operations and 9 pop operations during the conversion from infix to postfix notation

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

Infix to postfix conversion is applied in:

1. Calculator software.
2. Programming language compilers.
3. Expression evaluation.
4. Mathematical software.
5. Spreadsheet programs.
6. Computer algebra systems.
7. Calculator hardware.
8. Query languages.
9. Expression parsing.
10. Scientific and engineering simulations.