Experiment No.2
Aim - Convert an Infix expression to Postfix expression using stackADT.
Name: Shravani Sandeep Raut
Roll No: 48
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>
#include <ctype.h>
#define MAX 100
int a[MAX];
int top = -1;

void push(int n)
{
        if(top == (MAX-1))
        {
                printf("Stack is full");
        }
        else
        {
                top++;
                a[top] = n;
        }
}

char pop()
{
   int m;
   if(top ==-1)
   {
        printf("Underflow");
```

```
        return -1;
    }
    else
    {
        m = a[top];
        top--;
        return m;
    }
}

int priority(char ch)
{
        if(ch == '^')
        {
                return 3;
        }
        if(ch == '*' || ch == '/' || ch == '%')
        {
                return 2;
        }
        if(ch == '+' || ch =='-')
        {
                return 1;
        }
        else
        {
                return 0;
        }
}


void Conversion(char postfix[])
{
        char ch,x;
        int i;
        for(i=0;postfix[i]!= '@'; i++)
        {
                ch = postfix[i];
                if(ch == '(')
                {
                        push(ch);
                }
                else if (ch ==')')
                {
                        while((x=pop()) != '(')
                        {
                                printf("%c", x);
                        }
                }
```

```
		else if(isalnum(ch))
		{
			printf("%c", ch);
		}
		else
		{
			while(priority(a[top])>= priority(ch))
			{
				printf("%c", pop());
			}
			push(ch);
		}
	}
	while(top != (-1))
	{
		printf("%c", pop());
	}
}



void main()
{
	int i;
	char postfix[100];
	printf("At the end of expression put @");
	for(i=0;i<=99 ; i++)
	{
		scanf("%c", &postfix[i]);
		if(postfix[i] =='@')
		{
			break;
		}
	}
	Conversion(postfix);
}
```

Output:

```
At the end of expression put @(a+b/c)@
abc/+
```

Conclusion:

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

To convert the infix expression **(A + (C / D)) * B** to postfix, follow these steps:

1. Start by converting the expression inside the parentheses: **C / D → C D /**
2. Now, replace **C / D** in the expression: **A + (C D /)**
3. Convert the remaining expression: **A + (C D /) → A C D / +**
4. Finally, multiply by **B**: **(A C D / +) * B → A C D / + B ***

So, the postfix expression is:
A C D / + B *

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

For the conversion of the infix expression (A + (C / D)) * B to postfix A C D / + B *, let's count the push and pop operations required based on the stack used during the conversion process:

(A + (C / D)) * B

Step-by-step conversion:

1. **'('**: Push to stack (push = 1).
2. **'A'**: Add to postfix (no push/pop).
3. **'+'**: Push to stack (push = 2).
4. **'('**: Push to stack (push = 3).
5. **'C'**: Add to postfix (no push/pop).
6. **'/'**: Push to stack (push = 4).
7. **'D'**: Add to postfix (no push/pop).
8. **')'**: Pop '/' from stack and add to postfix (pop = 1), then pop '(' (pop = 2).
9. **'+'**: Pop '+' from stack and add to postfix (pop = 3).
10. **'*'**: Push to stack (push = 5).
11. **'B'**: Add to postfix (no push/pop).
12. **End**: Pop '*' from stack and add to postfix (pop = 4).

Total operations:

- **Push operations**: 5
- **Pop operations**: 4

So, **5 push** and **4 pop** operations were required.

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

1. **Expression evaluation** by computers and calculators for easy processing.
2. **Compiler design** for generating intermediate code.
3. **Interpreters** to efficiently evaluate expressions.
4. **Stack-based calculators** for simplifying arithmetic operations.
5. **Assembly language** to avoid handling operator precedence.