Experiment No.2
Convert an Infix expression to Postfix expression using stack ADT.
Name:Umang Borse
Roll No:03
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:

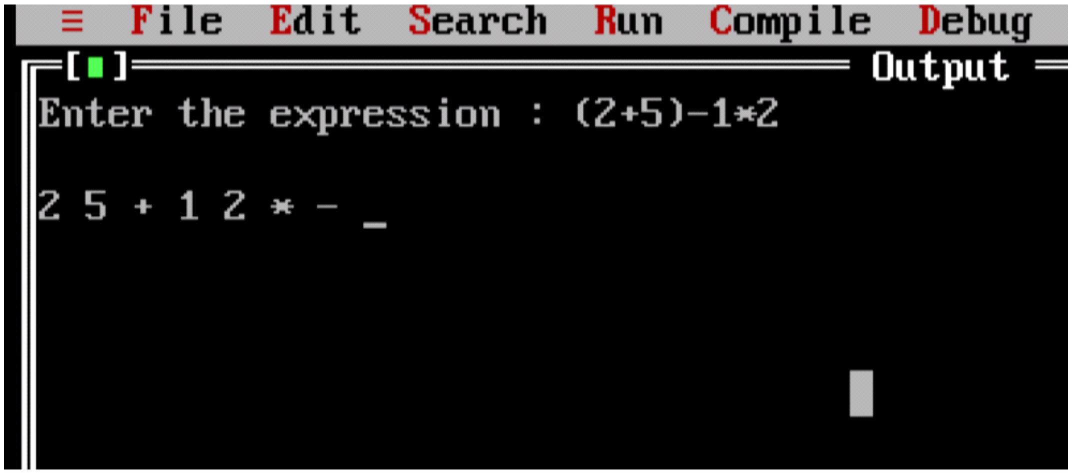
```
 #include<stdio.h>
#include<ctype.h>
char stack[100];
int top = -1;
void push(char x)
{
 stack[++top] = x;
}
char pop()
{
 if(top == -1)
 return -1;
 else
 return stack[top--];
}
int priority(char x)
{
 if(x == '(')
 return 0;
 if(x == '+' || x == '-')
 return 1;
 if(x == '*' || x == '/')
 return 2;
 return 0;
}
```

```
int main()
{
 char exp[100];
 char *e, x;
 printf("Enter the expression : ");
 scanf("%s",exp);
 printf("\n");
 e = exp;

 while(*e != '\0')
 {
 if(isalnum(*e))
 printf("%c ",*e);
 else if(*e == '(')
 push(*e);
 else if(*e == ')')
 {
 while((x = pop()) != '(')
 printf("%c ", x);
 }
 else
 {
 while(priority(stack[top]) >= priority(*e))
 printf("%c ",pop());
 push(*e);
 }
 e++;
 }

 while(top != -1)
 {
 printf("%c ",pop());
 }return 0;
}
```

Output:



```
Enter the expression : (2+5)-1*2

2 5 + 1 2 * - _
```

Conclusion:

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

"A + (C / D) * B" to postfix notation

1. Start with an empty stack and an empty output string.

2. Scan the infix expression from left to right:

   - A is an operand, so it goes to the output string: "A"

   - + is an operator, push it onto the stack.

   - (is an opening parenthesis, push it onto the stack.

   - C is an operand, goes to the output string: "A C"

   - / is an operator, push it onto the stack.

   - D is an operand, goes to the output string: "A C D"

   -) is a closing parenthesis: Pop operators from the stack and add them to the output until the opening parenthesis is encountered. We get "A C D /" in the output string.

   - * is an operator, push it onto the stack.

   - B is an operand, goes to the output string: "A C D / B"

3. At the end, pop any remaining operators from the stack and add them to the output string. In this case, we pop the + operator and add it to the output.

The resulting postfix expression is "A C D / B * +."

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

In the conversion of the infix expression "A + (C / D) * B" to postfix notation using the Shunting Yard Algorithm, the number of push and pop operations required can be counted as follows:

1. Push Operations:

   - Push "+" onto the stack.

   - Push "(" onto the stack.

   - Push "/" onto the stack.

   - Push "*" onto the stack.

   Total Push Operations: 4

2. Pop Operations:

   - Pop "+" (when encountering ")").

   Total Pop Operations: 1

The number of push operations (4) and pop operations (1) required for the conversion of this specific expression is as mentioned above.

Where is the infix to postfix conversion used or applied?

The conversion of infix expressions to postfix notation is applied in:

1. Expression evaluation and simplification.

2. Calculator software and reverse polish notation (RPN) calculators.

3. Parsing, compilers, and code generation.

4. Validation of arithmetic expressions.

5. Computer algebra systems and symbolic mathematics.

6. Handling logical and Boolean expressions.

7. Certain algorithms that require postfix notation as input.