Experiment No.2
Convert an Infix expression to Postfix expression using stack ADT.
Name: Singh Rahul Rammilan
Roll No:56
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:

- Understand the use of Stack.
- Understand how to import an ADT in an application program.
- Understand the instantiation of Stack ADT in an application program.
- 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:

<u>Conversion of infix to postfix</u>

```
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
        • Repeatedly pop from stack and add it to the postfix
          expression until a "(" is encountered.
        • Discard the "(". That is, remove the "(" from
          stack and do not add it to the postfix expression
      IF an operator 0 is encountered, then
        • 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
        • 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:

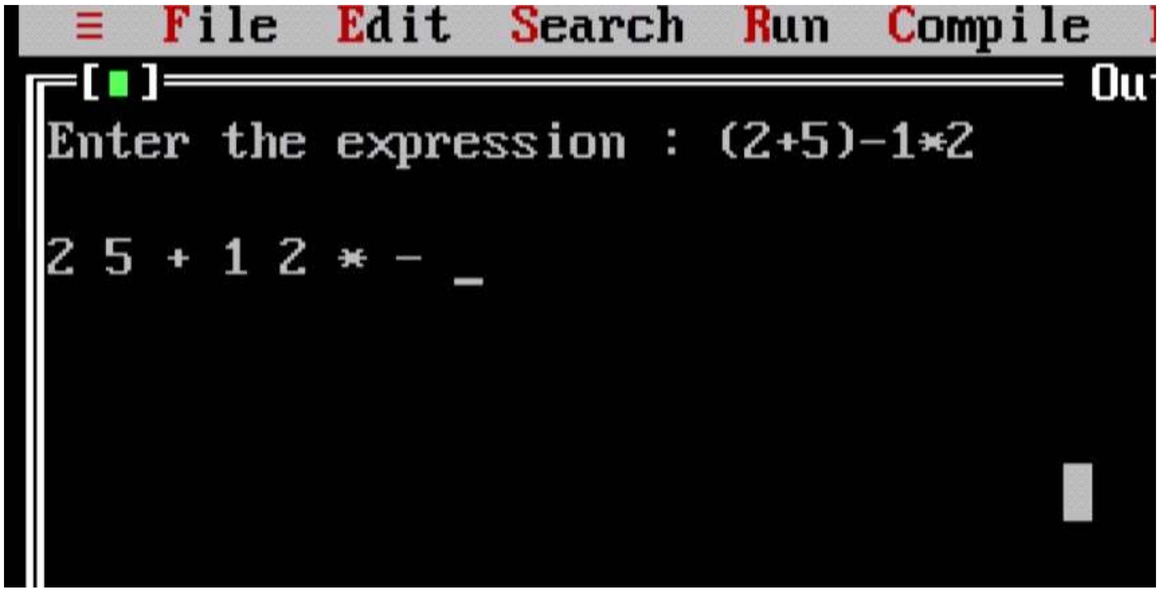
```
#inclu
de<std
io.h>
#inclu
de<cty
pe.h>
char
stack[
100];
int top
= -1;
void push(char x)
{
 stack[++top] = x;
```

```
}
char pop()
{
 if(top == -1)

 r
 e
 t
 u
 r
 n
 -
 1
 ;
 e
 l
 s
 e
 return stack[top--];
}
int priority(char x)
{

 i
 f
 (
 x
 =
 =
 '
 (
 '
 )
 r
 e
 t
 u
 r
 n
 0
 ;
 if(x ==
 '+' || x
 == '-')
 return
 1;
 if(x ==
 '*' || x
 == '/')
 return
 2;
 return 0;
}
int main()
{
 ch
```

```
ar
ex
p[1
00]
;
ch
ar
*e,
x;
printf("Enter the
expression : ");
scanf("%s",exp);
p
r
i
n
t
f
(
"
\
n
"
)
;
e
=
e

x

p

;

while(*e != '\0')

{

if(isaln

um(*e

))

printf(

"%c ",

*e);

else

if(*e

==

'(')

pus

h(*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
- The given infix expression "A+(C/D))B" is converted to postfix
  as follows: "ACD/+B".

- How many push and pop operations were required for the above conversion?
- To convert the expression, you can count the push and pop operations:
  - Push operations: 6
  - Pop operations: 6

- Where is the infix to postfix conversion used or applied?
- Infix to postfix conversion is used or applied in various computer

science and programming areas, including:

- Compiler Design: It's used to convert mathematical expressions into a format that can be easily evaluated by a computer.
- Expression Parsing: Converting infix expressions to postfix makes it easier to perform operations and calculations in the correct order.
- Calculator Applications: In postfix notation, you don't need parentheses, making it suitable for calculator input.
- Stack-Based Algorithms: Postfix notation is well-suited for stack-based algorithms and data structures, making it useful in computer science and data processing.