NAME: GAURAV KISHOR PATIL

DIV: 2 ROLL NO:54

BATCH: C

Experiment No.2
Convert an Infix expression to Postfix expression using stack ADT.
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 writing algebraic expressions without the use of parentheses or rules of operator precedence. The expression (A+B)/(C–D) would be written as AB+CD-/ in postfix notation. An expression is scanned from user in infix form; it is converted into postfix form and then evaluated without considering the parenthesis and priority of the operators.

An arithmetic expression consists of operands and operators. For a given expression in an postfix form, stack can be used to evaluate the expression. The rule is whenever an operands comes into the string push it on to the stack and when an operator is found then last two elements from the stack are poped and computed and the result is pushed back on to the stack. One by one whole string of postfix expression is parsed and final result is obtained at an end of computation that 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*+

So, the Postfix Expression is 23*21-/53*+

Algorithm :

Conversion

1. Read the symbol one at a time from the input expression.
2. If it is operand, output it.
3. If it is opening parenthesis, push it on stack.
4. If it is an operator, then check its incoming priority
    5. If stack is empty, push operator on stack.
    6. If the top of stack is opening parenthesis, push operator on stack
    7. If it has higher priority operator than the top of stack, push operator on stack.
    8. Else pop the operator from the stack and output it, repeat step 4
9. If it is a closing parenthesis, pop operators from stack and output them until an opening parenthesis is encountered. pop and discard the opening parenthesis.
10. If there is more input go to step 1
11. If there is no more input, pop the remaining operators to output.

Code:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#define max 20
char stack[max],infix[max],postfix[max];
int top=-1;
void push(char);
char pop();
int is_empty();
void convert();
int priority(char);

int main()
{
 top=-1;
 gets(infix);
 convert();
 printf("%s",postfix);
return 0;

}
void push(char c)
{
top++;
stack[top]=c;
```

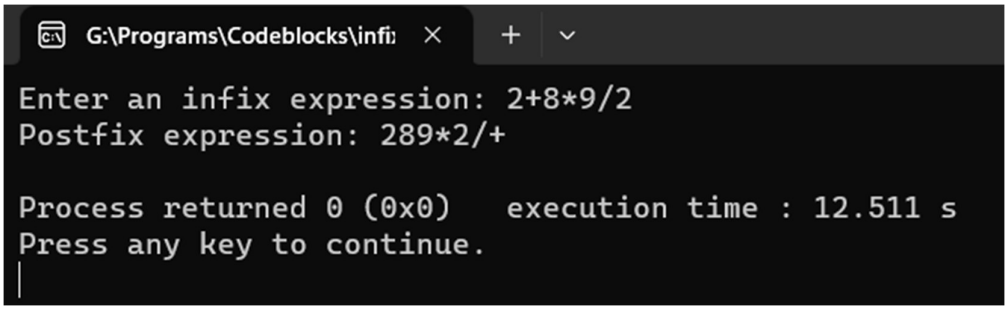
```
}
char pop()
{
   char c;
if(is_empty()==1)
{
   printf("underflow");


}
else {
   c=stack[top];
   top--;
   return c;
}
}

int is_empty()
{
   if(top==-1)
   return 1;
   else return 0;


}
void convert(){
char symbol,next;
int i=0,j=0;
```

```
symbol=infix[i];
for(i=0;i<strlen(infix);i++)
{
 switch(symbol)
 {
 case '(': push(symbol);
 break;
 case ')':
 while((next=pop())!='(')
 postfix[j++]=next;
break;
 case '+':
   case '-':
       case '*':
   case '/':
   case '^':
   while(!is_empty()&& priority(stack[top])>=priority(symbol))
   postfix[j++]=pop();
 push(symbol);
       break;
       default :
       postfix[j++]=symbol;

 }
```

```
}

while(!is_empty())
{
   postfix[j++]=pop();
}

postfix[j]='\0';

}
int priority(char symbol)
{
   switch (symbol)
   {
   case '+':
   case '-':
   return 1;
      break;
      case '*':
   case '/':
   return 2;
      break;

   case '^':
   return 3;
      break;
```

```
    default:
    return 0;
        break;
    }
}
```

Output:



Conclusion:

Using stack implementation and converting infix expressions to postfix expressions is a fundamental operation in computer science and programming. It's often used in compilers, calculators, and various applications that involve evaluating mathematical expressions. The conversion process involves changing the order of operators and operands in a way that makes the expression easier to evaluate.We here manipulate the operators rather than the operands and carry out the function.