EXPERIMENT NO 2

~ ARYA RAUL

COMPS 20

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 function of an ADT are accessed in an application program

Theory: Infix to postfix conversion is a fundamental operation in computer science and is commonly used in expression evaluation and parsing. It involves converting an infix expression (where operators are placed between operands) into a postfix expression (also known as Reverse Polish Notation or RPN, where operators are placed after their operands).

Infix: Infix notation is a common way of writing mathematical expressions or operations where the operators are placed between the operands. For example, in the expression "2 + 3," the operator "+" is placed between the operands "2" and "3." This is the typical way we write mathematical expressions and equations.

Postfix: Postfix notation, also known as Reverse Polish Notation (RPN), is a mathematical notation in which operators are placed after their operands. In other words, in postfix notation, you write the expression by first listing the operands and then specifying the operation to be performed. For example, the infix expression "2 + 3" would be written as "2 3 +" in postfix notation. This notation eliminates the need for parentheses to indicate the order of operations and is often used in calculators and computer programs for efficient expression evaluation.

Algorithm: Converting an infix expression to a postfix expression involves rearranging the operators and operands to their postfix notation. This process is typically done using a stack data structure. Here's an algorithm to convert an infix expression to a postfix expression:

Initialize an empty stack to hold operators.

Initialize an empty list to hold the postfix expression.

Scan the infix expression from left to right:

a. If the current token is an operand (operand can be a single digit/letter or a multi-digit/letter identifier), append it to the postfix expression list.

b. If the current token is an operator ('+', '-', '*', '/', etc.):

i. Pop operators from the stack and append them to the postfix expression list until: - The stack is empty. - The top of the stack is an opening parenthesis '('. - The current operator has lower precedence than the operator at the top of the stack, or they have equal precedence but the current operator is left-associative.

ii. Push the current operator onto the stack.

c. If the current token is an opening parenthesis '(', push it onto the stack.

d. If the current token is a closing parenthesis ')': i. Pop operators from the stack and append them to the postfix expression list until an opening parenthesis '(' is encountered and popped. ii. Discard the opening parenthesis from the stack.

After scanning the entire infix expression, pop any remaining operators from the stack and append them to the postfix expression list.

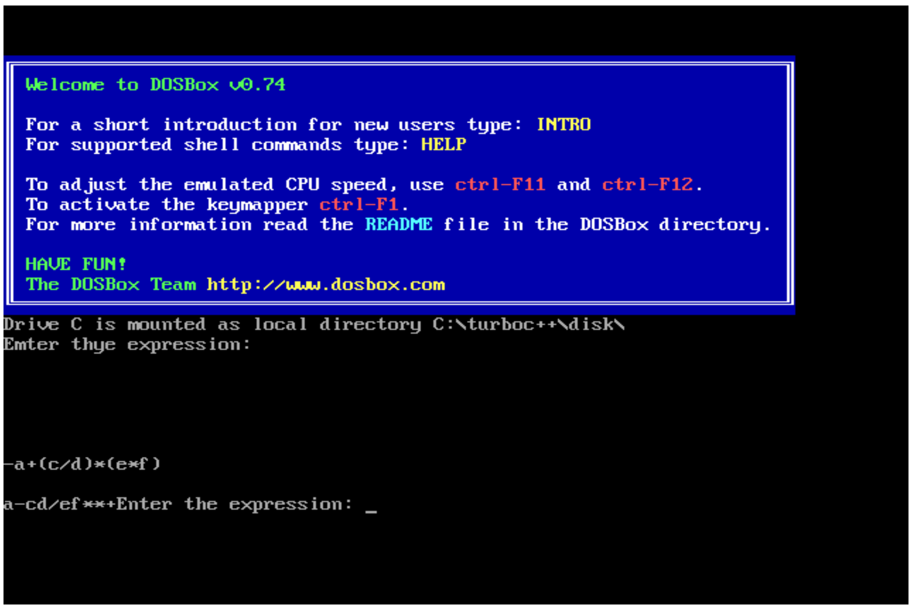
Code:

```
 #include<stdio.h>
#include<conio.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:



```
Welcome to DOSBox v0.74

For a short introduction for new users type: INTRO
For supported shell commands type: HELP

To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.

HAVE FUN!
The DOSBox Team http://www.dosbox.com
Drive C is mounted as local directory C:\turboc++\disk\
Emter thye expression:



-a+(c/d)*(e*f)

a-cd/ef**+Enter the expression: _
```

Conclusion: In conclusion, the infix to postfix conversion algorithm showcases the versatility and power of stack data structures in solving problems related to expression manipulation. By following the algorithm's steps, we can convert complex infix expressions into a format that is easier to evaluate and process, contributing to various applications in compilers, calculators, and more.