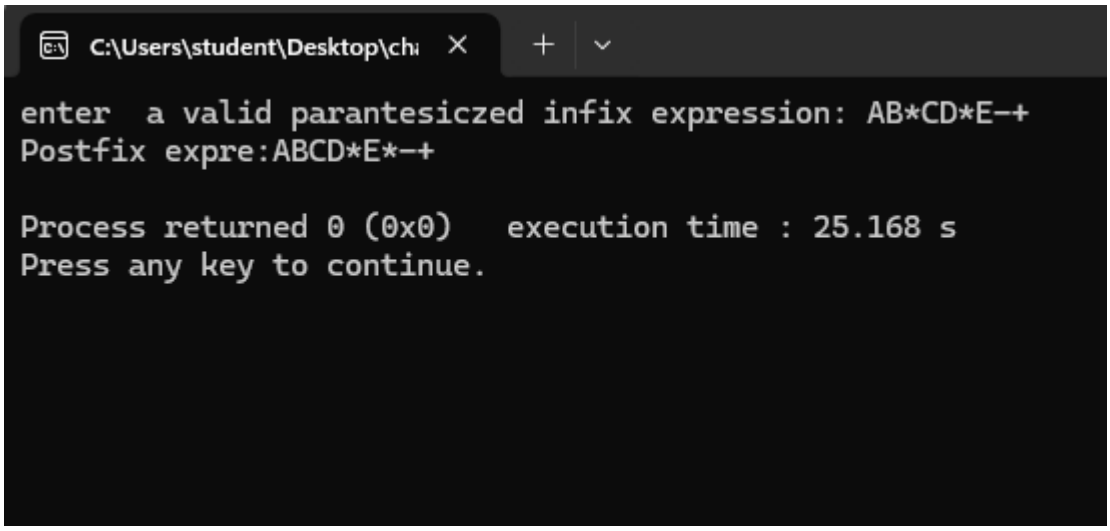
INFIX TO POSTFIX:

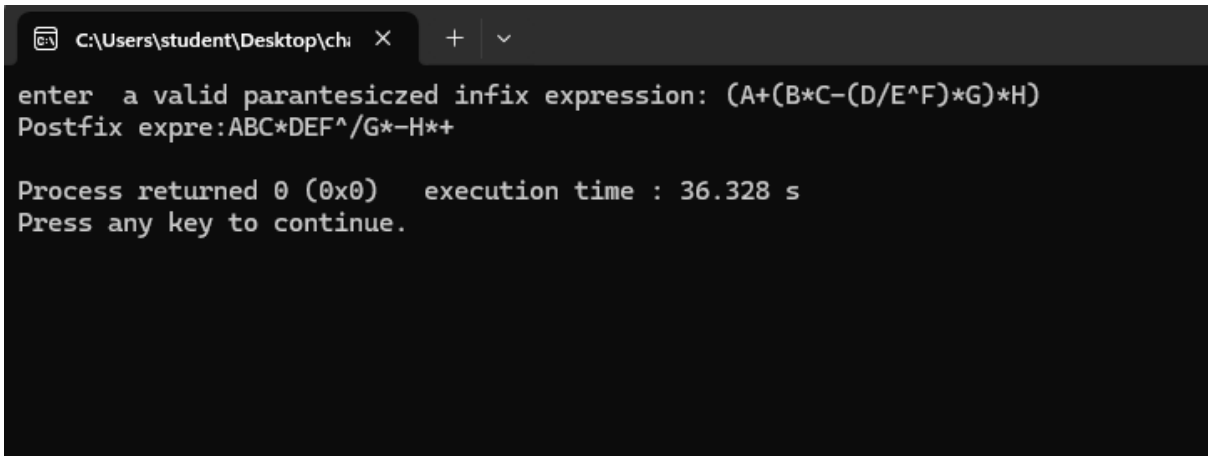
```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
# define MAX 100
char stack[MAX];
int top=-1;
void push(char c){
   if(top==MAX -1){
      printf("Stack Overflow\n");
      return ;
   }
   stack[++top]=c;
}
char pop(){
   if(top==-1){
      printf("stack Underflow\n");
      return -1;
   }
   return stack[top--];
}
char peek(){
   if(top==-1)return -1;
   return stack[top];
}
int precedence(char op){
   switch (op){
      case '+':
```

```
        case '-':
                return 1;
        case '*':
        case '/':
                return 2;
        case '^':return 3;
        case '(':
                return 0;

    }
    return -1;
}
int associativity(char op){
    if(op=="^")
      return  1;
    return 0;
}
void infixTopostfix(char infix[],char postfix[]){
    int i,k=0;
    char c;
    for (i=0;infix[i]!='\0';i++){
        c=infix[i];
        if (isalnum(c)){
            postfix[k++]=c;
        }
        else if(c=='('){
            push(c);
        }
        else if(c==')'){
            while (peek()!='('){
                postfix[k++]=pop();
```

```
        }
        pop();
      }
      else{
        while (top !=-1&&((precedence(peek()) > precedence(c))||(precedence(peek()) ==
precedence(c))&& associativity(c)==0)){
            postfix[k++]=pop();
        }
        push(c);
      }
   }
   while (top!=-1)
   {postfix[k++]=pop();
      /* code */
   }
   postfix[k]='\0';

}
int main() {
   char infix[MAX],postfix[MAX];
   printf("enter  a valid parantesiczed infix expression: ");
   scanf("%s",infix);
   infixTopostfix(infix,postfix);
   printf("Postfix expre:%s\n",postfix);
   return 0;

}
```

OUTPUT:





OBSERVATION:

Date __/__/__
Page _____

6/10/25

2. WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide).

=> Pseudocode

Infix to postfix

1 Start
2 ~~Initialize~~ Input an expression from the user
3 Create an empty stack for operators
4 Create an empty postfix string.
5 Read the infix expression from left to right
6 For each character:
   -> If character is an operand, add it to postfix.
   -> If character is '(', push it into the stack.
   -> If character is ')', pop & add to postfix until '(' is found, then remove '('
   -> If character is an operator (+, -, *, /):
      * pop & add to postfix while the top of stack has higher precedence or equal precedence
      * push the current operator to stack.
7 After reading all characters or when you reach the end of the expression, pop all the operators & add it to postfix exp.
8 print the postfix expression.
9 Stop.

A+B

6/10

Date __/__/__
Page _____

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>

#define Max 100

char stack[MAX];
int top = -1;

void push(char c) {
    if (top == MAX - 1) {
        printf("Stack Overflow\n");
        return;
    }
    stack[++top] = c;
}

char pop() {
    if (top == -1) {
        printf("Stack Underflow\n");
        return -1;
    }
    return stack[top--];
}

char peek() {
    if (top == -1) return -1;
    return stack[top];
}
int precedence(char op) {
  switch(op) {
      case '+':
      case '-':
          return 1;
```

```
        case '*':
        case '/':
            return 2;
        case '^':
            return 3;
        case '(':
            return 0;
    }
    return -1;
}

// Function to return associativity
// 0 = Left to right, 1 = Right to left.
int associativity (char op) {
    if (op == '^')
        return 1;    // right to left
    return 0;        // +, -, *, / -> left to right
}

// Function to convert infix to postfix
void infixtopostfix (char infix[], char
                     postfix[])
{
    int i, k = 0;
    char c;
    for (i = 0; infix[i] != '\0'; i++)
    {
        c = infix[i];
        if (isalnum(c)) {
            // operand -> directly to postfix
            postfix[k++] = c;
        }
```

```
else if (c == '(')
{
    push(c);
}
else if (c == ')')
{
    while (peek() != '(')
    {
        postfix[k++] = pop();
    }
    pop();    // discard '('
}
else
{
    // operator
    while (top != -1 &&
        ((precedence(peek()) > precedence(c))
        || (precedence(peek()) == precedence(c) && associativity(c) == 0)))
    {
        // L-to-R
        postfix[k++] = pop();
    }
    push(c);
}
}

// Pop remaining operators
while (top != -1) {
    postfix[k++] = pop();
}
postfix[k] = '\0';
}
```

Date __/__/__
Page _____

```
int main()
{
    char infix[MAX], postfix[MAX];
    printf("Enter a valid parenthesized
        infix expression: ");
    scanf("%s", infix);

    infixtopostfix(infix, postfix);
    printf("Postfix Expression: %s\n",
        postfix);
    return 0;
}
```

Output:-

Enter a valid parenthesized infix expression:
A*B+C*D-E

Postfix Expression: AB*CD*E-+

Enter a valid parenthesized infix expression:
(A+(B*C-(D/E^F)*G)*H)

Postfix Expression: ABC*DEF^/G*-H*+