WAP to convert parenthesized infix to postfix expressions. The expression consists of single character operands and binary operators. (+, -, *, /).

```
#define MAX 10
char stack[MAX];
int top = -1;

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

char pop()
{
    int a = stack[top];
    top--;
    return a;
}

char peek()
{
    return stack[top];
}

int precedence(char op)
{
    switch(op)
    {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        case '^':
            return 3;
        case '(':
            return 0;
    }
}

int associativity(char op)
{
    return 1; // '^'
    return 0;
}
```

```
void InfixToPostfix(char infix[], char postfix[])
{
    int j = 0;
    char c;
    for(i = 0; infix[i] != '\0'; i++)
    {
        c = infix[i];
        if(isalnum(c))
            postfix[j++] = c;
        else if(c == '(')
            push(c)
        else if(c == ')')
            while(peek() != '(')
                postfix[j++] = pop();
            pop()
        else
            while(top != -1 && ((precedence(peek()) > precedence(c)) ||
                (precedence(peek()) == precedence(c)) &&
                    associativity(c) == 0)))
                postfix[j++] = pop();
            push(c);
        while(top != -1)
            postfix[j++] = pop();
        postfix[j] = '\0';
    }
}

int main()
{
    char infix[MAX], postfix[MAX];
    scan infix;
    InfixToPostfix(infix, postfix);
    print postfix;
}
```

Sneha SB 14/10/25

```c
#include<stdio.h>
#include<ctype.h>
#include<string.h>
#define MAX 10
char stack[MAX];
int top=-1;
void push(char c)
{
    if (top==MAX-1)
        printf("stack overflow\n");
    else
    {
        top++;
        stack[top]=c;
    }
}
char pop()
{
    if(top==-1)
    {
        printf("stack underflow\n");
        return '\0';
    }
    else
    {
        printf("removed data=%c\n",stack[top]);
        char a = stack[top];
        top--;
        return a;
    }
}
char peek()
{
    if (top==-1)
    {
        printf("stack underflow\n");
        return '0';
    }
    else
        return stack[top];
}
```

```c
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)
{
    while(op=='^')
        return 1; // r-l
    return 0; // l-r
}
void infixtoPostfix(char infix[], char postfix[])
{
    int j=0;
    char c;
    for(int i=0; infix[i]!='\0'; i++)
    {
        c=infix[i];
        if(isalnum(c))
            postfix[j++]=c;
        else if(c=='(')
            push(c);
        else if(c==')')
        {
            while(peek()!='(')
                postfix[j++]=pop();
            pop(); // peek()='('
        }
```

```
        else
        {
            while (top!=-1 && (precedence(peek()) >
                precedence(c) || (precedence(peek()) ==
                precedence(c) && associativity(c)==0)))
                    postfix[j++] = pop();
            push(c); // others
        }
    }
    while (top!=-1)  // pop all data
        postfix[j++] = pop();
    postfix[j] = '\0';
}
int main()
{
    char infix[MAX], postfix[MAX];
    printf("enter the infix expression:\n");
    scanf("%s", infix);
    infixToPostfix(infix, postfix);
    printf("postfix expression: %s\n", postfix);
    return 0;
}
```

Output:

```
enter the infix expression:
(a+b)*c
removed data = +
removed data = (
removed data = *
postfix expression: ab+c*
```

o/p seen