Que. WAP to convert a given valid parenthesized infix arithmetic exp. to postfix expression. the expression consists of single character operands and the binary operators

+ , - , * , /

Write the pseudocode in your obser:

```
⇒ Infix to postfix (exp)
  {
     ~~exp~~ create a stack S
     for i <= 0 to length(exp)-1
     {
        if exp[i] is operand
           res <= res + exp[i]
        else if exp[i] is operator
           while ( ! S.empty() && HasHigherPre (S.top(), exp[i]))
           {
              res <= res + S.top()
              S.pop()
           }
           S.push (exp[i])
        elseif IsOpening Parentheses (exp[i])
           S.push (exp[i]);

        else if Is Closing Parentheses (exp[i])
        {
           while ( ! S.empty() && ! IsOpening Parentheses (S.top()))
           {
              res <= res + S.top()
              S.pop()
           }
           S.pop()
     }  }
while (!s.empty())
    {
            res <= res + s.top()
                    s.pop()
    }
    return res
}
```

[illegible] 14/10/25

• Code :

```
# 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;

    } else {

        return stack [ top ];

    }

}

int prec ( 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;

    } else {

        return 0;

    }

}
```

```
Void    infixToPostfix ( char infix[], char postfix []){
    int   k=0;
    Char  c;
    for ( int i = 0;  infix [], Char postfix []) {
            c = infix [i];
            if (isalnum (c)) {
                    postfix [k++] = c;
            } else if ( c== '(' ) {
                    push ( c);
        } else if  ( (c== ')') {
            while    (peek () != '(' && top != -1) {
                postfix [k++] = pop ();
            }
            if ( peek () == '(') {
                pop ();
            }
    } else {
        while ( top != -1 && ((prec (peek()) > prec (c))||
            (prec (peek()) == prec (c) && associativity
                                               (c)==0)))
        {
            postfix [k++] = pop ();
        }
        push (c);
    }   }
}
```

```
    while ( top !=-1 ) {
            postfix [K++ ]= pop();
        }
    postfix [ K ] = '\0';
}

int main () {
    char infix [MAX], postfix [MAX];
    printf (" enter valid expression :\n");
    scanf ("%s", infix);
    infixToPostfix (infix, postfix);
    printf (" postfix expression is : %s \n", postfix);
    return 0;
}
```

Output:

```
Enter valid expression :
(a+b )/c - u *v
postfix expression is : ab+c/uv*-
```

O/P

[illegible]