INFIX TO POSTFIX,PREFIX

NAME : ATHRESH KUMAR LABDE
RA1911033010146
M1

AIM: To verify INFIX to POSTFIX,PREFIX by implementing it in the code.

ALGORITHM:
Step 1: If the scanned character is an operand, put it into postfix expression.
Step 2: If the scanned character is an operator and operator's stack is empty, push the operator into operators' stack.
Step 3: If the operator's stack is not empty, there may be following possibilities.
If the precedence of scanned operator is greater than the top most operator of operator's stack, push this operator into operator 's stack.
If the precedence of scanned operator is less than the top most operator of operator's stack, pop the operators from operator's stack until we find a low precedence operator than the scanned character.
If the precedence of scanned operator is equal then check the associativity of the operator. If associativity left to right then pop the operators from stack until we find a low precedence operator. If associativity right to left then simply put into stack.
If the scanned character is opening round bracket (**'('**), push it into operator's stack.
If the scanned character is closing round bracket (**')'**), pop out operators from operator's stack until we find an opening bracket (**'('**).
Repeat Step 1,2 and 3 till expression has character
Step 4: Now pop out all the remaining operators from the operator's stack and push into postfix expression.
Step 5: Exit.

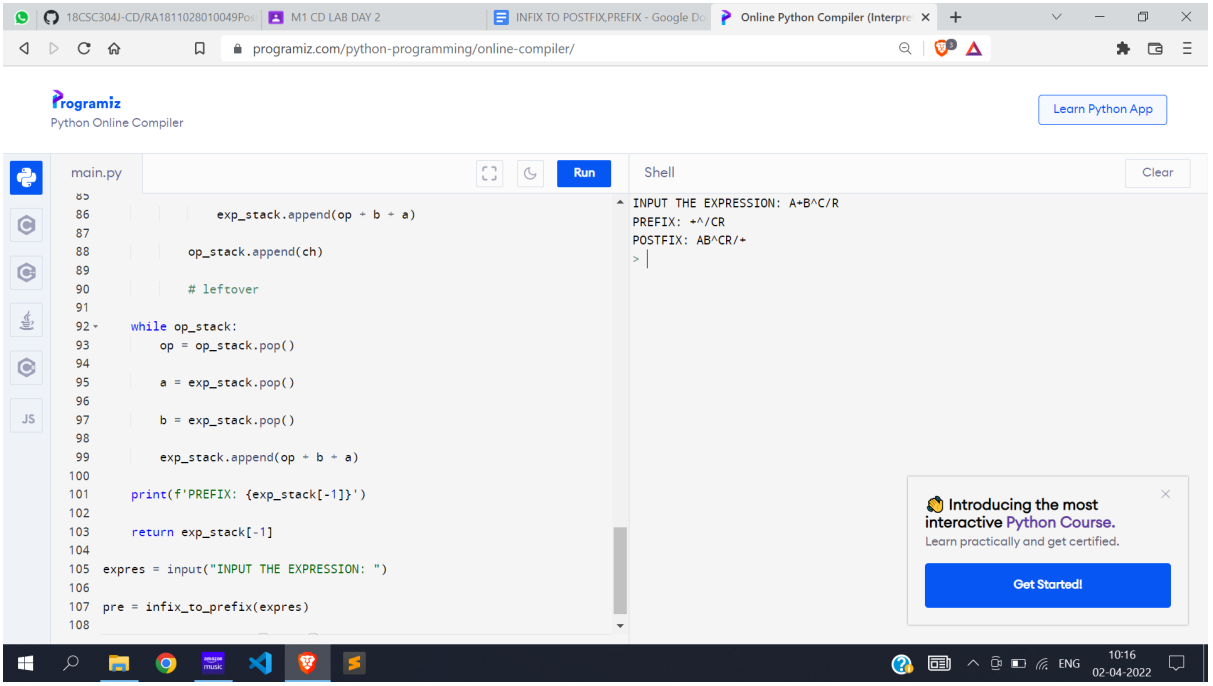
CODE :

```
OPERATORS = set(['+', '-', '*', '/', '(', ')'])

PRI = {'+': 1, '-': 1, '*': 2, '/': 2}

### INFIX ===> POSTFIX ###



def infix_to_postfix(formula):
```

```
stack = []  # only pop when the coming op has priority
output = ''
for ch in formula:
    if ch not in OPERATORS:
        output += ch
    elif ch == '(':
        stack.append('(')
    elif ch == ')':
        while stack and stack[-1] != '(':
            output += stack.pop()
        stack.pop()  # pop '('
    else:
        while stack and stack[-1] != '(' and PRI[ch] <= PRI[stack[-1]]:
            output += stack.pop()
        stack.append(ch)
        # leftover
while stack:
    output += stack.pop()
print(f'POSTFIX: {output}')
return output
```

```
### INFIX ===> PREFIX ###
def infix_to_prefix(formula):
    op_stack = []
    exp_stack = []
    for ch in formula:
        if not ch in OPERATORS:
            exp_stack.append(ch)
        elif ch == '(':
            op_stack.append(ch)
        elif ch == ')':
            while op_stack[-1] != '(':
                op = op_stack.pop()
                a = exp_stack.pop()
                b = exp_stack.pop()
                exp_stack.append(op + b + a)
            op_stack.pop()  # pop '('
        else:
            while op_stack and op_stack[-1] != '(' and PRI[ch] <= PRI[op_stack[-1]]:
                op = op_stack.pop()
                a = exp_stack.pop()
```

```
                b = exp_stack.pop()
                exp_stack.append(op + b + a)
            op_stack.append(ch)
            # leftover
    while op_stack:
        op = op_stack.pop()
        a = exp_stack.pop()
        b = exp_stack.pop()
        exp_stack.append(op + b + a)
    print(f'PREFIX: {exp_stack[-1]}')
    return exp_stack[-1]
expres = input("INPUT THE EXPRESSION: ")
pre = infix_to_prefix(expres)
pos = infix_to_postfix(expres)
```



RESULT : Hence the result is verified and implemented in form of the code.