18CSC304J/ Compiler Design

Submitted By:- ANANNYA P. NEOG (RA1911003010367)

Exp-10: Intermediate Code Generation - Postfix, Prefix

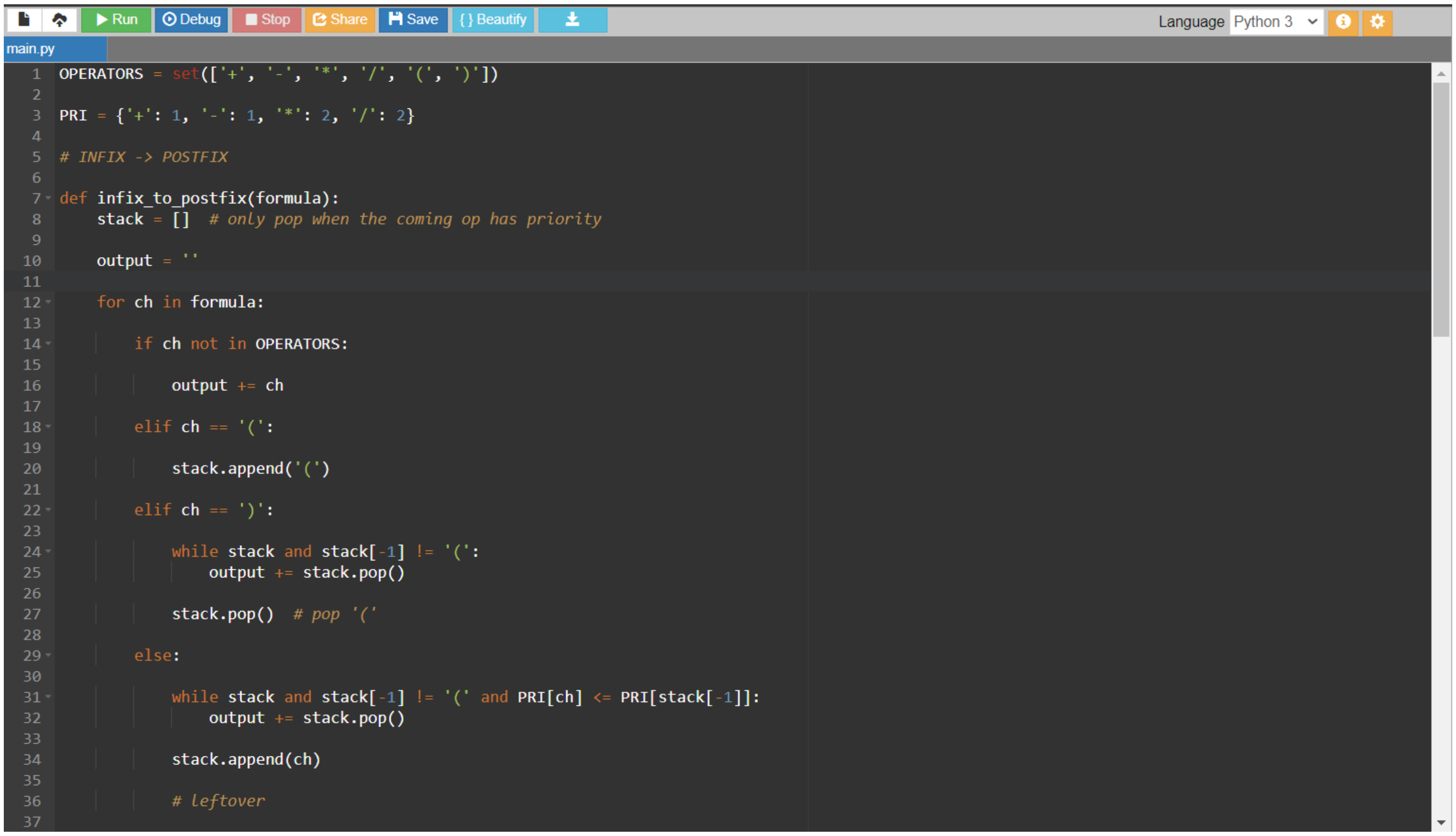
Aim:- To write code for Intermediate Code Generation - Postfix, Prefix

Codes:-

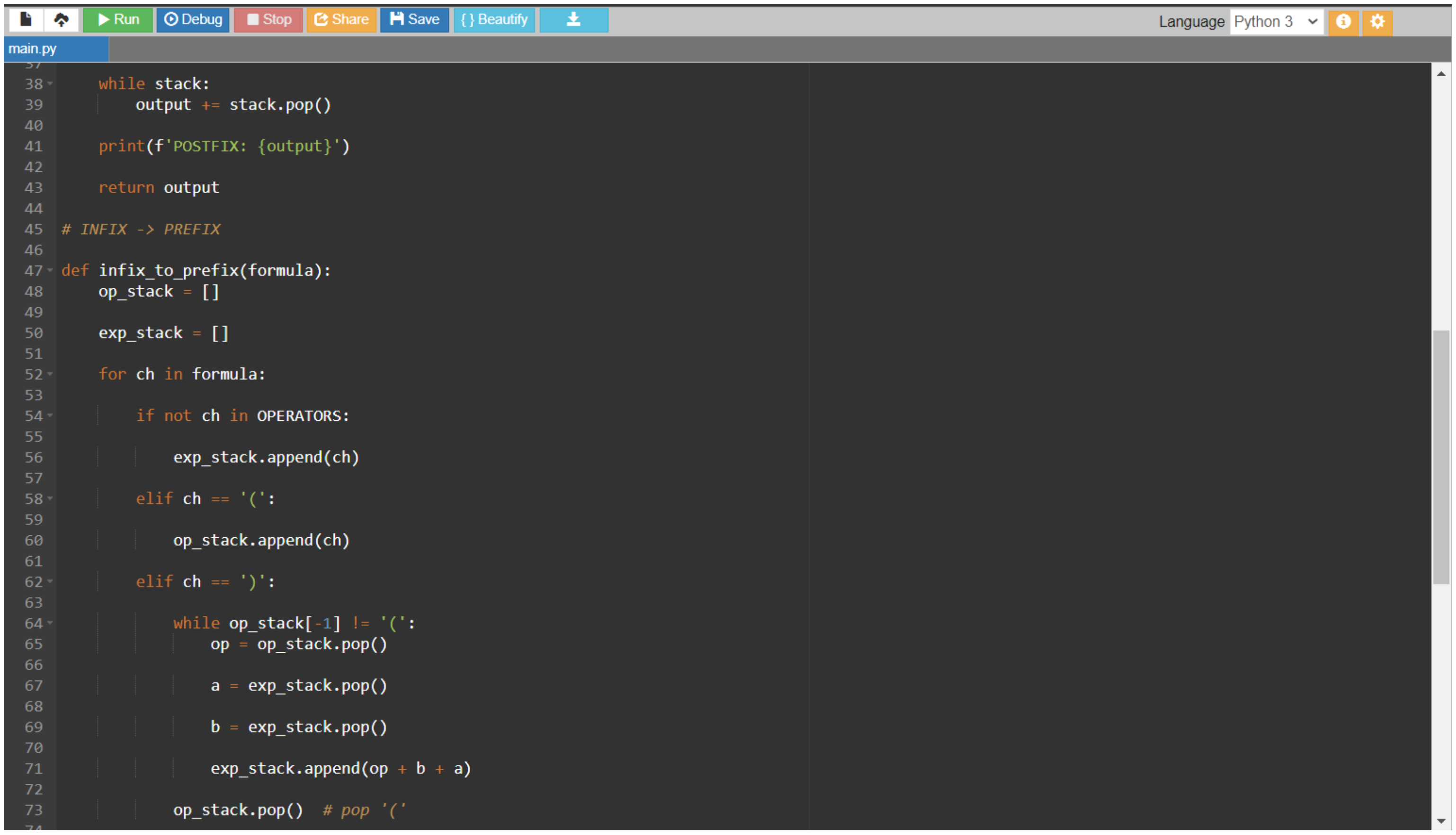
```
OPERATORS = set(['+', '-', '*', '/', '(', ')'])

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)

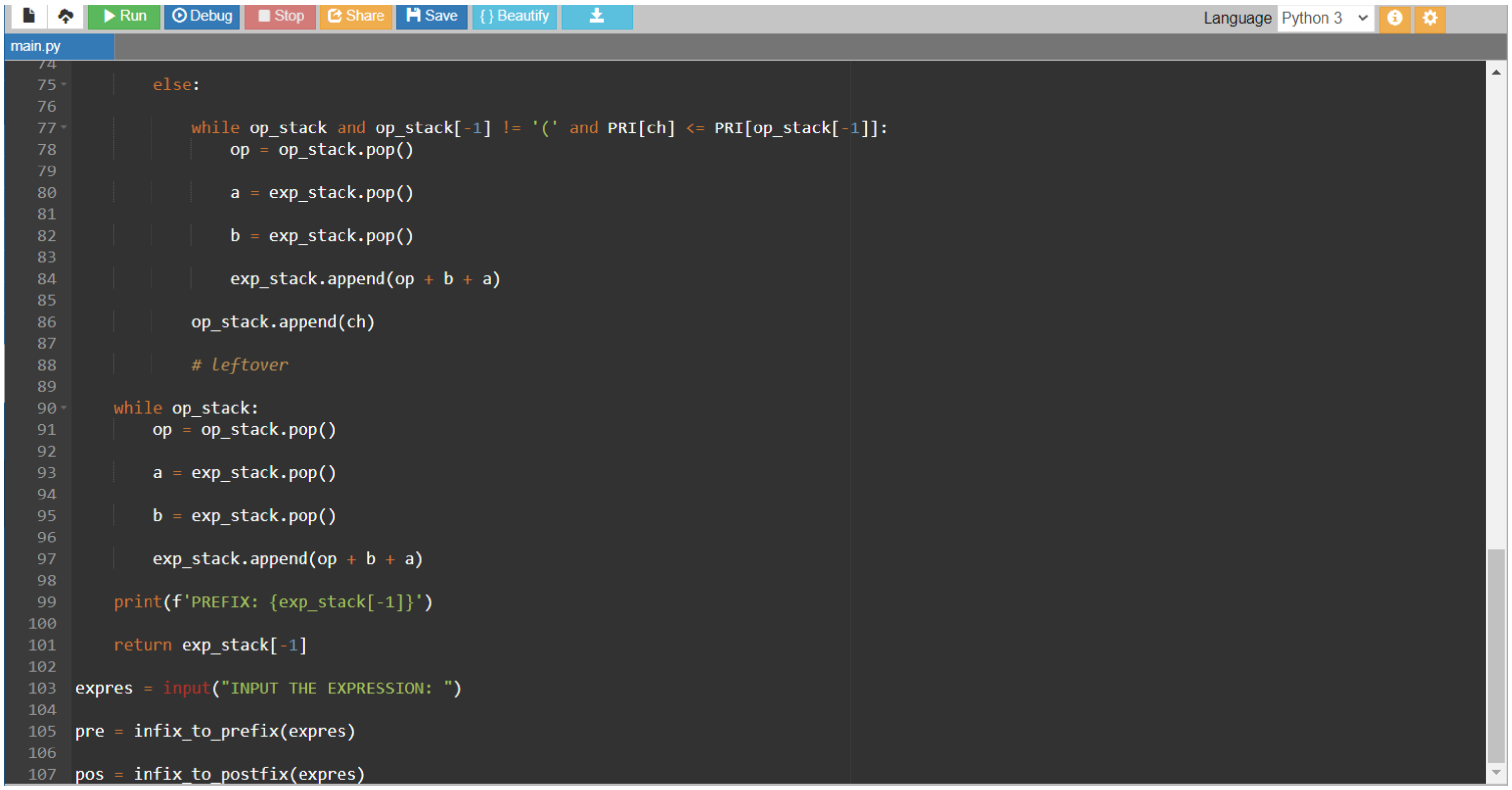
pos = infix_to_postfix(expres)



```python
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

```



```python
    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 '('
```



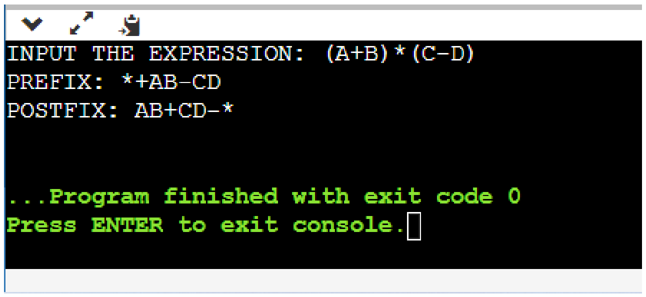
```python
        else:

            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)
```

Output:-

i) When input is:

(A+B)*(C-D)

as shown in the following output

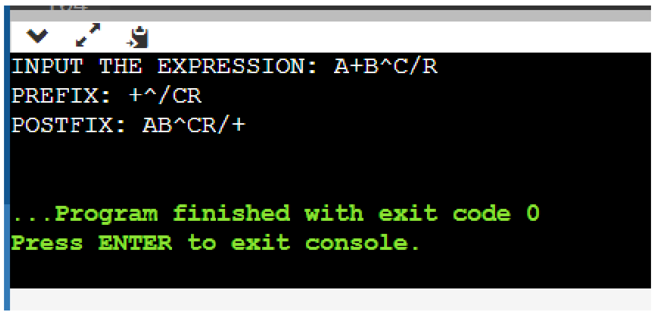


```
INPUT THE EXPRESSION: (A+B)*(C-D)
PREFIX: *+AB-CD
POSTFIX: AB+CD-*


...Program finished with exit code 0
Press ENTER to exit console.
```

i) When input is:

A+B^C/R

as shown in the following output



```
INPUT THE EXPRESSION: A+B^C/R
PREFIX: +^/CR
POSTFIX: AB^CR/+


...Program finished with exit code 0
Press ENTER to exit console.
```