Week -8

Q1

AIM: write a program to implement infix to postfix expression.

Description:

Infix expression: The expression of the form a op b. When an operator is in-between every pair of operands. Postfix expression: The expression of the form a b op. When an operator is followed for every pair of operands. To convert infix expression to postfix expression, we will use the stack data structure. By scanning the infix expression from left to right, when we will get any operand, simply add them to the postfix form, and for the operator and parenthesis, add them in the stack maintaining the precedence of them.

Program:

```
class Conversion:
    def __init__(self,capacity):
        self.capacity=capacity
        self.top=-1
        self.precedence={'+':1,'-':1,'*':2,'/':2,'^':3}
        self.array=[]
        self.output=[]
    def peek(self):
        if self.top!=-1:
            return self.array[-1]
    def pop(self):
        if self.top!=-1:
            self.top-=1
            return self.array.pop()
    def push(self,data):
        self.top+=1
        self.array.append(data)
```

```
    def notgreater(self, i):
        try:
            a = self.precedence[i]
            b = self.precedence[self.peek()]
            return True if a  <= b else False
        except KeyError:
            return False

    def infixtopostfix(self,exp):
        for i in exp:
            if i.isalpha():
                self.output.append(i)
            elif i=='(':
                self.push(i)
            elif i==')':
                #print(*self.array)
                while(self.top!=-1 and self.peek()!='('):

                    #print(">",a)
                    self.output.append(self.pop())
                if (self.top!=-1 and self.peek()=='('):
                    self.pop()


            else:
                while(self.top!=-1 and self.notgreater(i)):

                    self.output.append(self.pop())
```

```
            self.push(i)
        while(self.top!=-1):
            self.output.append(self.pop())
        print("Postfix expression is :")
        print("".join(self.output))
exp = input("Enter infix expression to be converted: ")
I = Conversion(len(exp))

I.infixtopostfix(exp)
```

Time complexity:

Worst case time complexity: O(n^2)

Average case time complexity: O(n^2)

Best case time complexity: O(n^2)

Output:

```
C:\Users\91630>python -u "c:\Users\91630\OneDrive\Deskto
Enter infix expression to be converted: (1+2)*3/7-8+3
Postfix expression is :
2+13+8-7/3*
```

Conclusion: The code is error free and runs as expected

Q2

AIM: write a program to evaluate postfix expression.

Description: The Postfix notation is used to represent algebraic expressions. The expressions written in postfix form are evaluated faster compared to infix notation as parenthesis are not required in postfix. We have discussed infix to postfix conversion.

Program:

```
class Postfix:
    def __init__(self):
        self.top=-1
        self.array=[]
    def push(self,d):
        self.top+=1
        self.array.append(d)
    def pop(self):
        if self.top!=-1:
            self.top-=1
            return self.array.pop()
    def evaluate(self,exp):
        for x in exp:
            if x.isdigit():
                self.push(x)
            else:
                A=self.pop()
                B=self.pop()
                self.push(str(eval(B+x+A)))
            #print(*self.array)
        return int(self.pop())
expression=input("Enter the postfix expression: ")
```

P=Postfix()

print("Result of Postfix exp entered after evaluation is ",P.evaluate(expression))

Time complexity: Time Complexity: O(n) where n is the size of the given string/expression

Output:

```
C:\Users\91630>C:/Users/91630/AppData/Local/Programs/Pyth
Enter the postfix expression: 987+-
Result of Postfix exp entered after evaluation is  -6
```

Conclusion: The code is error free and runs as expected.