Name : Rohit Kudache USN : 1BM18CS083

Sem : V Section : B Department = CSE

Lab test : AI-LAB TEST - 2 Date : 01/01/2021

```
combinations = [(True, True, True), (True, True, False),
(True, False, True), (True, False, False), (False, True, True),
(False, True, False), (False, False, True), (False, False, False)]
variable = { 'P' : 0, 'Q' : 1, 'R' : 2 }
    kb = ''
    q = ''
priority = {'~' : 3, 'v' : 1, '^' : 2}
def input_rules():
    global kb, q; kb = (input ("Enter rule: "))
    q = input ("Enter the query"):
def entilment():
    global kb, q
    print ("Truth Table Reference")
    print ('kb', 'alpha')
    for comb in combinations:
        s = evaluatePostfix (toPostfix (kb), comb)
        f = evaluate Postfix (topostfix (q), comb)
    print (s, f)
    print ('-' *10)
    if s and not f:
            return False:
    return True
def isOperand (c):
    return c.isalpha () and c != 'v'
def isLeftParanthesis (c):
    return c == '('
def isRightParanthesis (c):
    return c == ')'
```


Rheeet

```
def isEmpty (stack):
    return len (stack) == 0
def peek(stack):
    return Stack [-1]
def hasLessOrEqualPriority (c1, c2):
    try :
        return priority [c1] <= priority [c2]
    except keyError :
        return False
def toPostfix (infix):
    stack = []
    postfix = ''
    for c in infix :
        if isoperand (c):
            postfix += c
        else :
            if isLeftParanthesis (c) :
                stack.append (c)
            elif isRightParanthesis (c):
                operator = stack.pop()
                while not leftParanthesis (operator):
                    postfix += operator
                    operator = stack.pop()
            else :
                while (not isEmpty (stack)) and
                    hasLessOrEqualPriority (c, peek (stack)):
                    postfix += stack.pop()
                stack.append (c)
    while (not isEmpty (stack)):
        postfix += stack.pop()
    return postfix
```

```
def evaluatePostfix (exp, comb):
    stack = []
    for i in exp:
        if isOperand (i):
            stack.append (comb [variable.[i]])
        elif i == '~'
            val1 = stack.pop()
            stack.append (not val1)
        else:
            val1 = stack.pop()
            val2 = stack.pop()
            stack.append (_eval (i, val2, val1))
    return stack.pop()
def _eval (i, val1, val2):
    if i == '^':
        return val2 and val1
# Test 1
    input_rules()
    ans = entailment()
    if ans:
        print ("KB entails Query")
    else:
        print ("KB Doesn't entail Query")
```


Result:-