Date: / /

Name : V.K Prithvik Aniketh
USN : 1BM18CS119
Class : 5C

Code:

```
variable = {'S': 0, 'T': 1, 'R': 2}
priority = {'~': 3, '^': 1, 'v': 2}.

    def eval (i, val1, val2):
        if i == '~':
            return val2 and val1
        return val2 or val1

def isOperand (c):
    return c.isalpha() and c != 'v'

def isLeftParanthesis (c):
    return c == '('

def isRightParanthesis (c):
    return c == ')'

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 isLeftParanthesis(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 :
    val 1 = Stack . pop ()
    val 2 : stack . pop ()
    Stack . append ( eval (i, val 2, val 1)
return stack . pop ()

def CheckEntailment ():
    kb = Cinput ("Enter the knowledge base : "))
    query = Cinput ("Enter the query : "))
    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]]
    postfix_kb = toPostfix (kb)
    postfix_q = toPostfix (query)
    for combinations in combinations : combination)
        eval_kb = evaluatPostfix (postfix_kb, notion
        eval_q = evaluatPostfix (postfix_q, combi
        print (combination, ": kb =", eval_kb, ": q =",
            eval_q ).
        if (eval_kb == True):
            if (eval_q == False):
                print ("doesnt entail")
                return False .
            print ("Entails").
if __name__ == "__main__":
    CheckEntailment ().
```