LAB Exam-2

Naman Agarwal
1BM18CS057

```
variable = { 'P':0 , 'Q': 1 , 'R':2 }
priority = { '~':3, '^':2 , 'v':1 }

def eval (i, val1, val2)
        if i == '^':
                return val2 and V1
        return val2 or val1

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

def ~~isL~~ hasLessOrEqualpriority ( c1,c2):
    try:
            return priority (c1) <= priority [c2]
    except:
            return False.

def toPostfix ( infix):
        stack = []
        postfix = ''
        for c in infix.
                if isOperand (c) :
                        postfix += c
                else
```

N. Agarwal

Naman Agarwal
1BM18CS057

```
if   c == '('

          stack. append (c)

elif   c == ')'

          operator = stack.pop().
          while not ~~isleft~~ operator != '('

          postfix += operator.
          operator = stack.pop()

else :
                       stack.__len__ != 0
          while( ~~not~~ ~~stack.len~~

          and hasLessOrEqualPriority (c, peak(stack))

                       postfix += stack. pop()

          stack.append (c)
                   __len__
     while (stack != 0 ):

               postfix += stack. pop ().

return postfix.
```

[illegible]

Naman Agarwal
1BM18CS057

```
def evaluate ( 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 = " . pop()

            stack.append (eval(i , val1, val2))

    return stack.pop()

def check()

    kb = (input ("Enter knowledge base ")

    query = (input( " Enter Query ")

    combinations = [[True , True , True ], [True, True, False]
              [ True, False ,True ], [True , False , False ]
              [False, True ,True ] , [False, True, False]
```

[illegible]

Naman Agarwal
1BM18CS057

[Fake, Fu, Tu], [Fake, Fake, Fak]]

```
postfix_kb = toPostfix (kb)
postfix_q = to Postfix (Query)

for combination in combinations :
    eval kb = evaluatePostfix ( postfix_kb,
                                combination ]
    eval_q = evaluatePostfix ( postfix_q,
                               combination ]

    print ( combination, evalkb, eval_q)

    if (eval_kb == True):
            if (evalq == False):
                    return False .

Return true .
```

[illegible]