VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"Jnana Sangama", Belgaum -590014, Karnataka.

ARTIFICIAL INTELLIGENCE

LAB REPORT

Submitted by

SAQUIB NAUSHAD(1BM19CS144)

Under the Guidance of

Dr. Manjunath

Associate Professor, BMSCE

in partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

In

COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING

(Autonomous Institution under VTU)

BENGALURU-560019

Oct-2021 toJan-2022

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Artificial Intelligence carried out by**, SAQUIB NAUSHAD(1BM19CS144)** who are Bonafede students of **B. M. S. College of Engineering.** It is in partial fulfilment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visveswaraya Technological University, Belgaum during the year 2021-2022. The Lab report has been approved and satisfies the academic requirements in respect of **ARTIFICIAL INTELLIGENCE (20CS5PCAIP)** work prescribed for the said degree.

SignatureoftheGuide	Signature of the HOD
Dr.Manjunath	Dr. UmadeviV
Associate Professor	AssociateProf. & Head,Dept.ofCSE
BMSCE,Bengalur	BMSCE, Bengaluru

Index

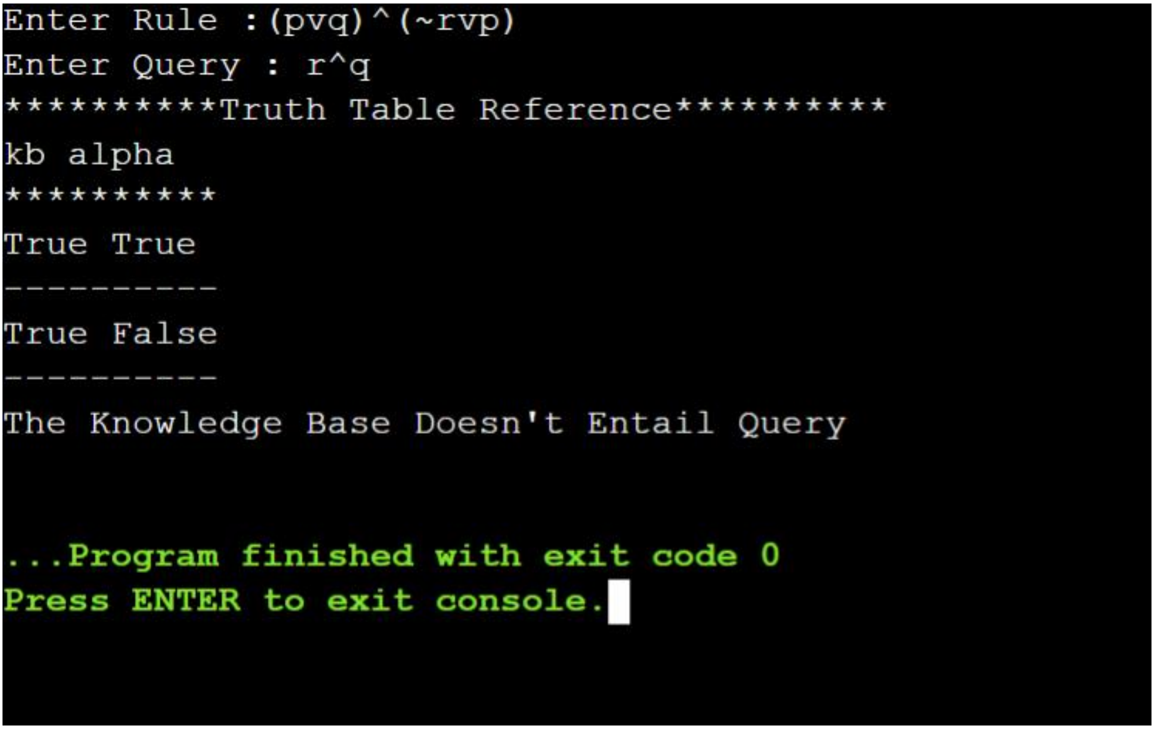
1. Create a knowledgebase using prepositional logic and show that the given query entails the knowledge base or not.

CODE:-

```
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("Knowledge base : "))
    q = input("Query : ")


def entailment():
    global kb, q
    print("*10+"Truth Table Reference"+"*10)
    print('kb    α')
    print('-'*10)
    for comb in combinations:
        s = evaluatePostfix(toPostfix(kb), comb)
        f = evaluatePostfix(toPostfix(q), comb)
        print(s,  f)
        if s is True and f is False:
            return False
    return True


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:
            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
    return val2 or val1


input_rules()
ans = entailment()
if ans:
    print("The Knowledge Base entails query")
    print(" KB |= α ")
else:
    print("The Knowledge Base does not entail query")
print("\n")
```

OUTPUT SCREEN[ALL OUTPUT ARE ALDREADY ON GITHUB]



```
Enter Rule :(pvq)^(~rvp)
Enter Query : r^q
**********Truth Table Reference**********
kb alpha
**********
True True
----------
True False
----------
The Knowledge Base Doesn't Entail Query


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

2. Create a knowledgebase using prepositional logic and prove the given query using resolution.

CODE:-

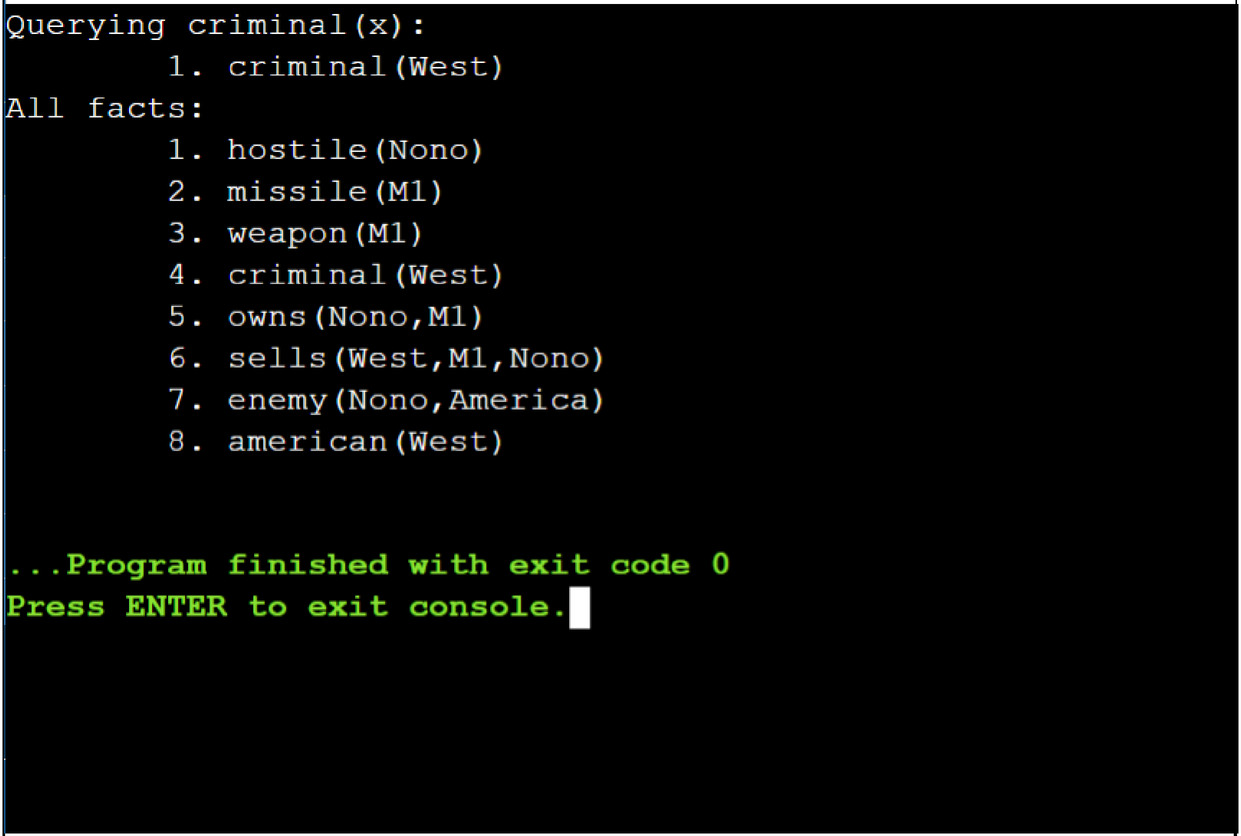
```
import re

def isVariable(x):
    return len(x) == 1 and x.islower() and x.isalpha()

def getAttributes(string):
    expr = '\([^)]+\)'
    matches = re.findall(expr, string)
    return matches

def getPredicates(string):
    expr = '([a-z~]+)\([^&|]+\)'
    return re.findall(expr, string)
class Fact:
    def __init__(self, expression):
        self.expression = expression
        predicate, params = self.splitExpression(expression)
        self.predicate = predicate
        self.params = params
        self.result = any(self.getConstants())

    def splitExpression(self, expression):
        predicate = getPredicates(expression)[0]
        params = getAttributes(expression)[0].strip('()').split(',')
        return [predicate, params]

    def getResult(self):
        return self.result

    def getConstants(self):
        return [None if isVariable(c) else c for c in self.params]

    def getVariables(self):
        return [v if isVariable(v) else None for v in self.params]

    def substitute(self, constants):
        c = constants.copy()
        f = f"{self.predicate}({','.join([constants.pop(0) if isVariable(p) else p for p in self.params])})"
        return Fact(f)
```

```
class Implication:
  def __init__(self, expression):
      self.expression = expression
      l = expression.split('=>')
      self.lhs = [Fact(f) for f in l[0].split('&')]
      self.rhs = Fact(l[1])

  def evaluate(self, facts):
      constants = {}
      new_lhs = []
      for fact in facts:
          for val in self.lhs:
              if val.predicate == fact.predicate:
                  for i, v in enumerate(val.getVariables()):
                      if v:
                          constants[v] = fact.getConstants()[i]
                  new_lhs.append(fact)
                   predicate, attributes = getPredicates(self.rhs.expression)[0],
           str(getAttributes(self.rhs.expression)[0])
      for key in constants:
          if constants[key]:
              attributes = attributes.replace(key, constants[key])
      expr = f'{predicate}{attributes}'
      return Fact(expr) if len(new_lhs) and all([f.getResult() for f in new_lhs]) else None

class KB:
  def __init__(self):
      self.facts = set()
      self.implications = set()

  def tell(self, e):
      if '=>' in e:
          self.implications.add(Implication(e))
      else:
          self.facts.add(Fact(e))
      for i in self.implications:
          res = i.evaluate(self.facts)
          if res:
              self.facts.add(res)

  def query(self, e):
      facts = set([f.expression for f in self.facts])
      i = 1
      print(f'Querying {e}:')
      for f in facts:
          if Fact(f).predicate == Fact(e).predicate:
              print(f'\t{i}. {f}')
              i += 1
```

```
    def display(self):
        print("All facts: ")
        for i, f in enumerate(set([f.expression for f in self.facts])):
            print(f'\t{i+1}. {f}')



#Test Case 1
kb = KB()
kb.tell('missile(x)=>weapon(x)')
kb.tell('missile(M1)')
kb.tell('enemy(x,America)=>hostile(x)')
kb.tell('american(West)')
kb.tell('enemy(Nono,America)')
kb.tell('owns(Nono,M1)')
kb.tell('missile(x)&owns(Nono,x)=>sells(West,x,Nono)')
kb.tell('american(x)&weapon(y)&sells(x,y,z)&hostile(z)=>criminal(x)')
kb.query('criminal(x)')
kb.display()
```

OUTPUT SCREEN

Test Case 1:



```
Querying criminal(x):
        1. criminal(West)
All facts:
        1. hostile(Nono)
        2. missile(M1)
        3. weapon(M1)
        4. criminal(West)
        5. owns(Nono,M1)
        6. sells(West,M1,Nono)
        7. enemy(Nono,America)
        8. american(West)


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

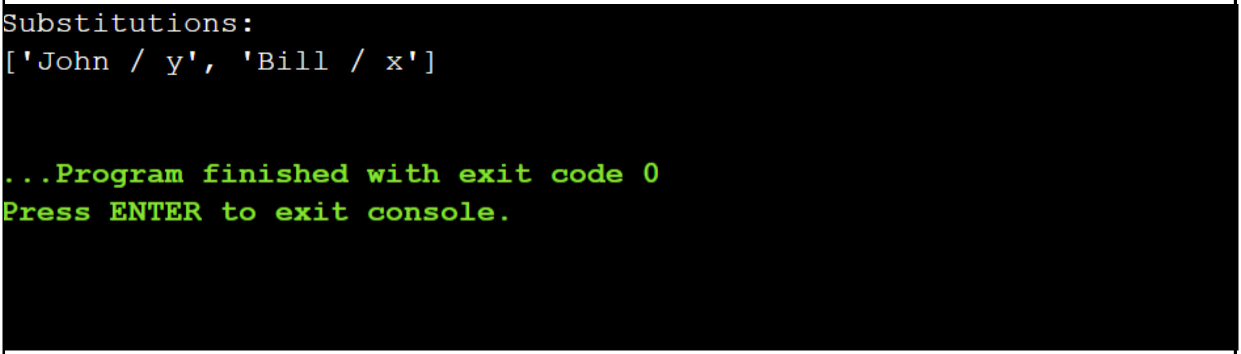
3. Implement unification in first order logic.

CODE:-

```
import re

def getAttributes(expr):
   expr = expr.split("(")[1:]
   expr = "(".join(expr)
   expr = expr[:-1]
   expr = re.split("(?<!\(.),(?!.\))", expr)
   return expr
def getInitialPredicate(expr):
   return expr.split("(")[0]

def isConstant(char):
    return char.isupper() and len(char) == 1

def isVariable(char):
    return char.islower() and len(char) == 1

def replaceAttributes(expr, old, new):
   attr = getAttributes(expr)
   for index, val in enumerate(attr):
       if val == old:
           attr[index] = new
   predicate = getInitialPredicate(expr)
   return predicate + "(" + ",".join(attr) + ")"

def apply(expr, subs):
   for sub in subs:
       new, old = sub
       expr = replaceAttributes(expr, old, new)
   return expr
def checkOccurs(var, expr):
   if expr.find(var) == -1:
       return False
   return True

def getFirstPart(expr):
   attr = getAttributes(expr)
   return attr[0]
```

```
def getRemainingPart(expr):
    predicate = getInitialPredicate(expr)
    attr = getAttributes(expr)
    newExpr = predicate + "(" + ",".join(attr[1:]) + ")"
    return newExpr
def unify(exp1, exp2):
    if exp1 == exp2:
        return []

    if isConstant(exp1) and isConstant(exp2):
        if exp1 != exp2:
            return False

    if isConstant(exp1):
        return [(exp1, exp2)]

    if isConstant(exp2):
        return [(exp2, exp1)]

    if isVariable(exp1):
        if checkOccurs(exp1, exp2):
            return False
        else:
            return [(exp2, exp1)]

    if isVariable(exp2):
        if checkOccurs(exp2, exp1):
            return False
        else:
            return [(exp1, exp2)]

    if getInitialPredicate(exp1) != getInitialPredicate(exp2):
        print("Cannot be unified")
        return False

    attributeCount1 = len(getAttributes(exp1))
    attributeCount2 = len(getAttributes(exp2))
    if attributeCount1 != attributeCount2:
        return False

    head1 = getFirstPart(exp1)
    head2 = getFirstPart(exp2)
    initialSub = unify(head1, head2)
    if not initialSub:
        return False
    if attributeCount1 == 1:
        return initialSub

    tail1 = getRemainingPart(exp1)
    tail2 = getRemainingPart(exp2)

    if initialSub != []:
        tail1 = apply(tail1, initialSub)
        tail2 = apply(tail2, initialSub)

    remainingSub = unify(tail1, tail2)
    if not remainingSub:
        return False

    initialSub.extend(remainingSub)
    res = []
    for tup in initialSub:
      st = ' / '.join(tup)
      res.append(st)
    return res
exp1 = "knows(John,x)"
exp2 = "knows(y,Bill)"
subs = unify(exp1, exp2)
print("Substitutions:")
print(subs)
```

OUTPUT SCREEN



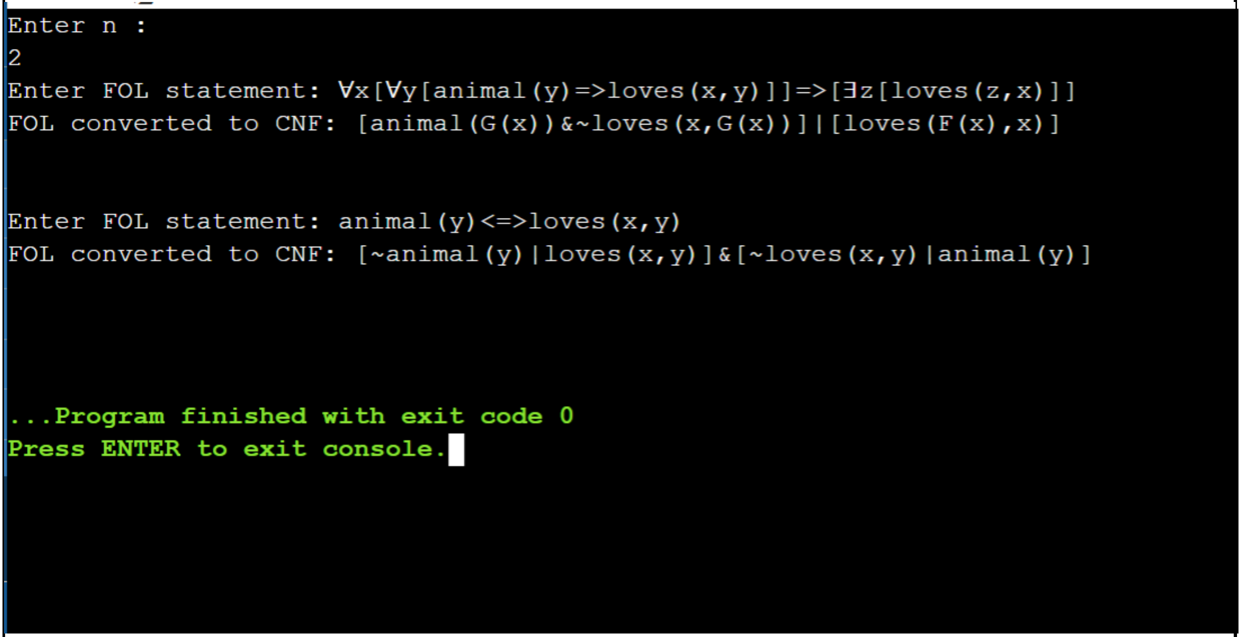
```
Substitutions:
['John / y', 'Bill / x']


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

4. Convert given first order logic statement into Conjunctive Normal Form (CNF).

CODE:-

```
def getAttributes(string):
    expr = '\([^)]+\)'
    matches = re.findall(expr, string)
    return [m for m in str(matches) if m.isalpha()]

def getPredicates(string):
    expr = '[a-z~]+\([A-Za-z,]+\)'
    return re.findall(expr, string)

def DeMorgan(sentence):
    string = ".join(list(sentence).copy())
    string = string.replace('~~',")
    flag = '[' in string
    string = string.replace('~[',")
    string = string.strip(']')
    for predicate in getPredicates(string):
        string = string.replace(predicate, f'~{predicate}')
    s = list(string)
    for i, c in enumerate(string):
        if c == '|':
            s[i] = '&'
        elif c == '&':
            s[i] = '|'
    string = ".join(s)
    string = string.replace('~~',")
    return f'[{string}]' if flag else string

def Skolemization(sentence):
    SKOLEM_CONSTANTS = [f'{chr(c)}' for c in range(ord('A'), ord('Z')+1)]
    statement = ".join(list(sentence).copy())
    matches = re.findall('[∀∃].', statement)
    for match in matches[::-1]:
        statement = statement.replace(match, ")
        statements = re.findall('\[\[[^]]+\]]', statement)
        for s in statements:
            statement = statement.replace(s, s[1:-1])
        for predicate in getPredicates(statement):
            attributes = getAttributes(predicate)
            if ''.join(attributes).islower():
                statement = statement.replace(match[1],SKOLEM_CONSTANTS.pop(0))
            else:
                aL = [a for a in attributes if a.islower()]
                aU = [a for a in attributes if not a.islower()][0]
                statement = statement.replace(aU, f'{SKOLEM_CONSTANTS.pop(0)}({aL[0] if
len(aL) else match[1]})')
    return statement
import re

def fol_to_cnf(fol):

    statement = fol.replace("<=>", "_")
    while '_' in statement:
        i = statement.index('_')
        new_statement = '[' + statement[:i] + '=>' + statement[i+1:] + ']&['+ statement[i+1:] + '=>'
+ statement[:i] + ']'
        statement = new_statement
    statement = statement.replace("=>", "-")
    expr = '\[([^]]+)\]'
    statements = re.findall(expr, statement)
    for i, s in enumerate(statements):
        if '[' in s and ']' not in s:
            statements[i] += ']'
    for s in statements:
        statement = statement.replace(s, fol_to_cnf(s))
    while '-' in statement:
        i = statement.index('-')
        br = statement.index('[') if '[' in statement else 0
        new_statement = '~' + statement[br:i] + '|' + statement[i+1:]
        statement = statement[:br] + new_statement if br > 0 else new_statement
    while '~∀' in statement:
        i = statement.index('~∀')
        statement = list(statement)
        statement[i], statement[i+1], statement[i+2] = '∃', statement[i+2], '~'
        statement = ''.join(statement)
    while '~∃' in statement:
        i = statement.index('~∃')
        s = list(statement)
        s[i], s[i+1], s[i+2] = '∀', s[i+2], '~'
        statement = ''.join(s)
    statement = statement.replace('~[∀','[~∀')
    statement = statement.replace('~[∃','[~∃')
    expr = '(~[∀|∃].)'
    statements = re.findall(expr, statement)
    for s in statements:
        statement = statement.replace(s, fol_to_cnf(s))
    expr = '~\[[^]]+\]'
    statements = re.findall(expr, statement)
    for s in statements:
        statement = statement.replace(s, DeMorgan(s))
    return statement

print("Enter n : ")
n = int(input())
while n:
 statement = input("Enter FOL statement: ")
 print(f"FOL converted to CNF: {Skolemization(fol_to_cnf(statement))} \n\n")
 n -= 1
```

OUTPUT SCREEN



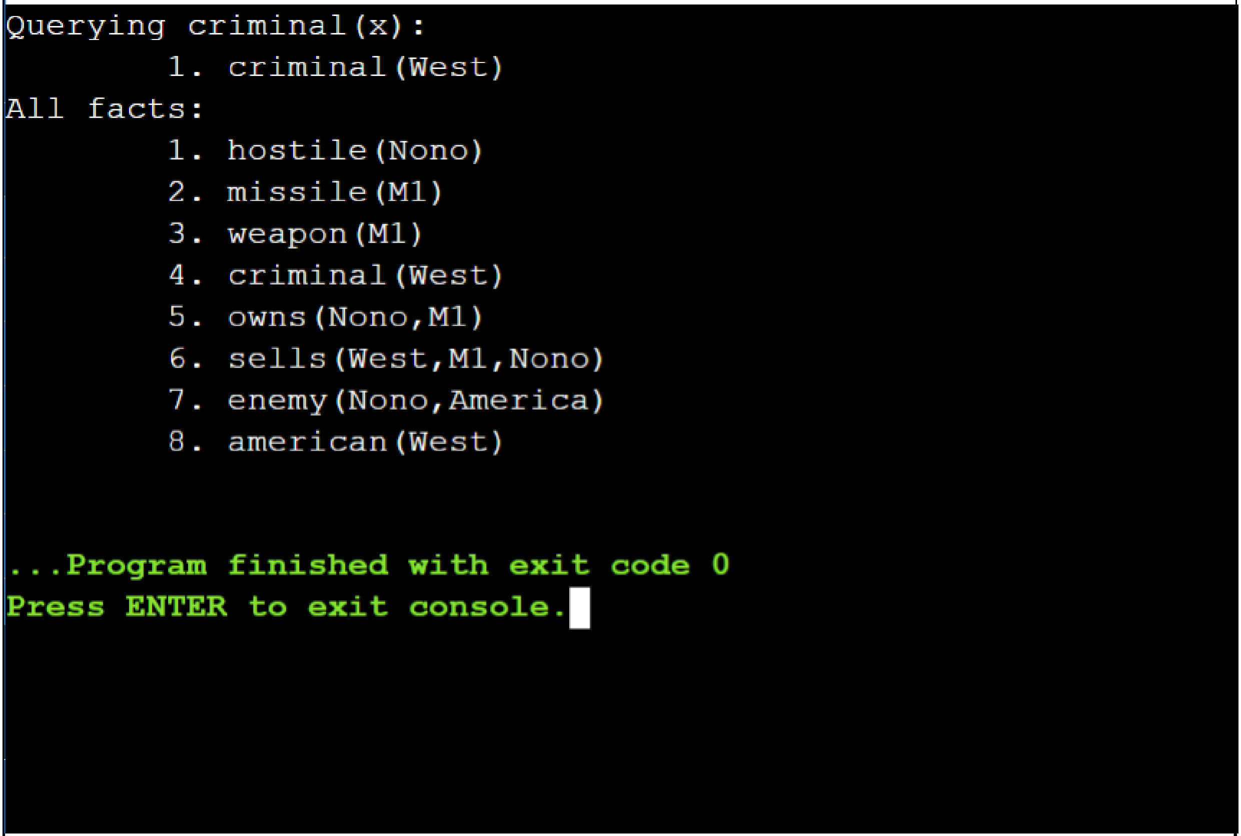
```
Enter n :
2
Enter FOL statement: ∀x[∀y[animal(y)=>loves(x,y)]]=>[∃z[loves(z,x)]]
FOL converted to CNF: [animal(G(x))&~loves(x,G(x))]|[loves(F(x),x)]


Enter FOL statement: animal(y)<=>loves(x,y)
FOL converted to CNF: [~animal(y)|loves(x,y)]&[~loves(x,y)|animal(y)]




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

5. Create a knowledgebase consisting of first order logic statements and prove the given query using forward reasoning.

CODE:-

```
import re

def isVariable(x):
    return len(x) == 1 and x.islower() and x.isalpha()

 def getAttributes(string):
    expr = '\([^)]+\)'
    matches = re.findall(expr, string)
    return matches

def getPredicates(string):
    expr = '([a-z~]+)\([^&|]+\)'
    return re.findall(expr, string)
class Fact:
    def __init__(self, expression):
        self.expression = expression
        predicate, params = self.splitExpression(expression)
        self.predicate = predicate
        self.params = params
        self.result = any(self.getConstants())

    def splitExpression(self, expression):
        predicate = getPredicates(expression)[0]
        params = getAttributes(expression)[0].strip('()').split(',')
        return [predicate, params]

    def getResult(self):
        return self.result

    def getConstants(self):
        return [None if isVariable(c) else c for c in self.params]

    def getVariables(self):
        return [v if isVariable(v) else None for v in self.params]

    def substitute(self, constants):
        c = constants.copy()
        f = f"{self.predicate}({','.join([constants.pop(0) if isVariable(p) else p for p in
self.params])})"
        return Fact(f)

class Implication:
    def __init__(self, expression):
        self.expression = expression
        l = expression.split('=>')
        self.lhs = [Fact(f) for f in l[0].split('&')]
        self.rhs = Fact(l[1])

    def evaluate(self, facts):
        constants = {}
        new_lhs = []
        for fact in facts:
            for val in self.lhs:
                if val.predicate == fact.predicate:
                    for i, v in enumerate(val.getVariables()):
                        if v:
                            constants[v] = fact.getConstants()[i]
                    new_lhs.append(fact)
        predicate, attributes = getPredicates(self.rhs.expression)[0],
str(getAttributes(self.rhs.expression)[0])
        for key in constants:
            if constants[key]:
                attributes = attributes.replace(key, constants[key])
        expr = f'{predicate}{attributes}'
        return Fact(expr) if len(new_lhs) and all([f.getResult() for f in new_lhs]) else None

class KB:
    def __init__(self):
        self.facts = set()
        self.implications = set()

    def tell(self, e):
        if '=>' in e:
            self.implications.add(Implication(e))
        else:
            self.facts.add(Fact(e))
        for i in self.implications:
            res = i.evaluate(self.facts)
            if res:
                self.facts.add(res)
    def query(self, e):
        facts = set([f.expression for f in self.facts])
        i = 1
        print(f'Querying {e}:')
        for f in facts:
            if Fact(f).predicate == Fact(e).predicate:
                print(f'\t{i}. {f}')
                i += 1

    def display(self):
        print("All facts: ")
        for i, f in enumerate(set([f.expression for f in self.facts])):
            print(f'\t{i+1}. {f}')


#Test Case 1
 kb = KB()
 kb.tell('missile(x)=>weapon(x)')
 kb.tell('missile(M1)')
 kb.tell('enemy(x,America)=>hostile(x)')
 kb.tell('american(West)')
 kb.tell('enemy(Nono,America)')
 kb.tell('owns(Nono,M1)')
 kb.tell('missile(x)&owns(Nono,x)=>sells(West,x,Nono)')
 kb.tell('american(x)&weapon(y)&sells(x,y,z)&hostile(z)=>criminal(x)')
 kb.query('criminal(x)')
 kb.display()
```

OUTPUT SCREEN



```
Querying criminal(x):
        1. criminal(West)
All facts:
        1. hostile(Nono)
        2. missile(M1)
        3. weapon(M1)
        4. criminal(West)
        5. owns(Nono,M1)
        6. sells(West,M1,Nono)
        7. enemy(Nono,America)
        8. american(West)


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