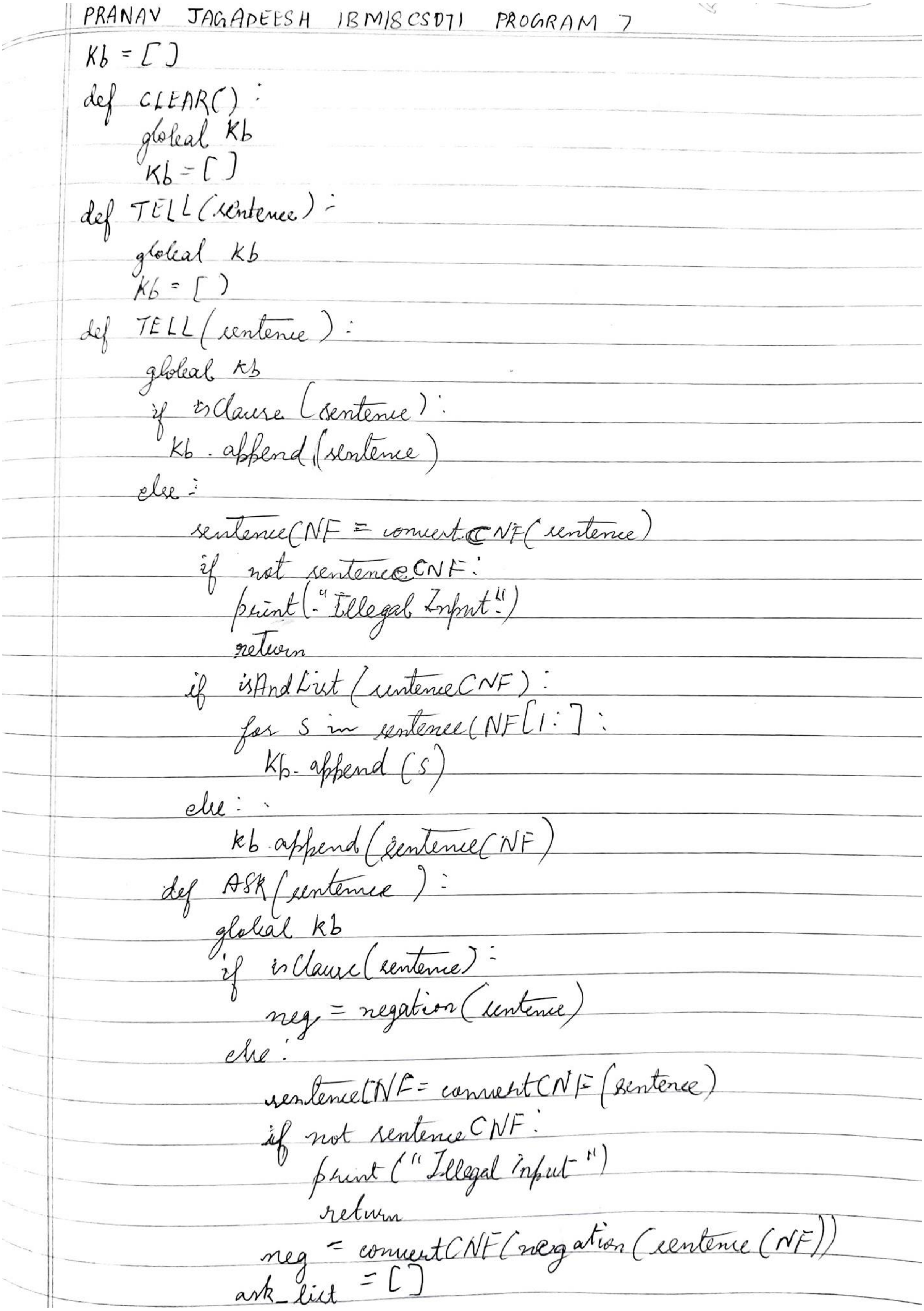
Program 7: Create a knowledgebase using prepositional logic and prove the given query using resolution. Write_up:



```
PRANAV JAGADEESH 1BM18CS071 PROGRAM 7
Kb = [ ]
def CLEAR( ):
    global Kb
    Kb = [ ]
def TELL(sentence):
    global kb
    Kb = [ )
def TELL(sentence):
    global kb
    if isClause(sentence):
      Kb.append(sentence)
    else:
        sentenceCNF = convertCNF(sentence)
        if not sentenceCNF:
          print("Illegal Input!")
          return
        if isAndList(sentenceCNF):
          for s in sentenceCNF[1:]:
            Kb.append(s)
        else:
          kb.append(sentenceCNF)
     def ASK(sentence):
       global kb
        if isClause(sentence):
          neg = negation(sentence)
        else:
          sentenceCNF = convertCNF(sentence)
          if not sentenceCNF:
             print("Illegal input")
             return
          neg = convertCNF(negation(sentenceCNF))
       ask_list = [ ]
```

```
if isANDList(neg):
    for n in neg[1:]:
        nCNF = makeCNF(n)
        if type(nCNF).__name__ == 'list':
            ask_list.insert(0, nCNF)
        else:
            ask_list.insert(0, nCNF)
  else:
        ask_list = [neg]
    clauses = ask_list + Kb[:]
    while True:
        new_clauses = []
        for c1 in clauses:
            for c2 in clauses:
                if c1 is not in c2:
                    resolved = resolve(c1, c2)
                    if. resolved == False:
                        continue
                    if resolved == []:
                        return True
                    new_clauses.append(resolved)
        if len(new_clauses) == 0:
            return False
        new_in_clauses = True
        for n in new_clauses:
            if n not in clauses:
                new_in_clauses = False
                clauses.append(n)
        if new_in_clauses:
            return False
        return False
```

```
def. resolve (arg_one, arg_two):
    resolved = False
    s1 = make_sentence (arg_one)
    s2 = make_sentence (arg_two)
    resolve_s1 = None
    resolve_s2 = None
    for i in s1:
        if isNotList (i):
            a1 = i[1]
            a1_not = True
        else:
            a1 = i
            a1_not = False
        for j in s2:
            if isNotList (j):
                a2 = j[1]
                a2_not = True
            else:
                a2 = j
                a2_not = False
            if a1 == a2:
                if a1_not != a2_not:
                    if resolved:
                        return False
                    else:
                        resolved = True
                        resolve_s1 = i
                        resolve_s2 = j
                        break
        if not resolved:
            return False
```

```
s1.remove (resolve_s1)
s2.remove (resolve_s2)
result = clear_duplicate (s1+s2)
 if len (result) == 1:
   return result [0]
 elif len (result) >1:
    result.insert (0, 'or')
  return result
  def make_sentence (arg):
      if isLiteral (arg) or isNotList(arg):
          return [arg]
      if isOrList(arg):
          return clear_duplicate (arg[1:])
      return
   def clear_duplicate (arg):
       result = []
       for i in range(0, len(arg)):
           if arg[i] not in (arg [i+1:]:
               result.append(arg[i])
       return result
   def isClause (sentence):
       if isLiteral (sentence):
           return True
       if isNotList (sentence):
           if isLiteral (sentence [1]):
               return True
           else:
               return False
       if isOrList (sentence):
           for i in range (1, len(sentence)):
               if len(sentence[i]) > 2:
                   return False
```

```
            elif not isClause(sentence[i]):
                return False
          return True
        return False
    #def negation(sentence):
     def isCNF(sentence):
         if isClause(sentence):
            return True
        elif isAndList(sentence):
              for s in sentence[1:]:
                  if not isClause(s):
                      return False
              return True
          return False
       def negation(sentence):
           if isLiteral(sentence):
              return['not", sentence]
          if isNotList[sentence):
              return sentence[1]
          if isANDList(sentence):
              result =['or')
              for i in sentence[1:]:
                  if isNotList(sentence):
                      result.append(i[1])
                  else:
                        result.append(['not', sentence])
                  return result
              if isOrList(sentence):
                  result =['and']
                  for i in sentence[1:]:
                      if isNotList(sentence):
                          result.append(i[1])
        else :
            result.append(['not', sentence ])
        return result
    if isORList (sentence):
        result = ['and ']
        for i in sentence [:]:
            if isNOTList (sentence ):
                result.append(i[1])
            else :
                result.append (['not', i])
        return result
    return None
def convert CNF (sentence) :
    while not isCNF(sentence) :
        if sentence is None :
            return None
        sentence = makeCNF(sentence )
    return sentence
def makeCNF (sentence ):
    if isLiteral(sentence) :
        return sentence
    if (type (sentence).__name__=='list'):
        operand = sentence [0]
        if isNOTList(sentence):
            if isLiteral (sentence[1]):
                return sentence
            cnf = makeCNF (sentence [1])
            if cnf [0] == 'not :
                return makeCNF(cnf [1])
            if cnf[0] == 'or':
                result = ['and']
                for i in range (1, len(cnf)):
```

```
        result.append(makeCNF(['not', cnf[i]]))
    . return result
    if cnf[0] == = 'and':
      result = ['or']
      for i in range(1, len(cnf)):
          result.append(make CNF(['not', cnf[i]]))
      return result
    return "False: not"
  if operand == 'implies' and len(sentence) == 3:
    return makeCNF(['or', ['not', makeCNF
    (sentence[1])], makeCNF(sentence[2])])
  if operand == 'biconditional' and len(sentence) == 3:
    s1 = make CNF(['implies', sentence[1], sentence[2]])
    s2 = makeCNF(['implies', sentence[2], sentence(1]])
    return makeCNF(['and', s1, s2])
  if isAndList(sentence):
    result = ['and']
    for i in range(1, len(sentence)):
        cnf = make CNF(sentence[i])
        if isAndList(cnf):
            for i in range(1, len(cnf)):
                result.append(make CNF(cnf[i]))
            continue
        result.append(makeCNF(cnf))
    return result
  if isOrList(sentence):
    result1 = ['or']
      for i in range(1, len sentence)):
        cnf = make CNF(sentence[i])
        if isOrList(cnf):
            for i in range(1, len(cnf)):
                result1.append(make CNF(cnf[i]))
        continue
    result1.append(makeCNF(cnf))
    while True :
        result2 = ['and']
        and_clause = None
        for r in result1 :
            if isAndList(r):
                and_clause = r
                break
        if not and_clause :
            return result1
        result1.remove(and_clause)
        for i in range (1, len(and_clause)):
            temp={'or', and_clause[i]]
            for o in result1[1:]:
                temp.append(makeCNF(o))
            result2.append(makeCNF(temp))
        result1 = makeCNF(result2)
    return None
return None
def isLiteral(item):
    if. type(item).__name__ =='str':
    #if len(item) == 2 :
        #if item[0] == 'not':
            return True
    return False
def isNotList (item):
        if type(item).__name__ =='list':
            if len(item)==2 :
                if item[0] == 'not':
                    retrun True
            return False
```

Date ____
Page ____

```
def isAndList(item):
        if type(item).__name__ =='list':
            if len(item) > 2:
                if item[0] =='and':
                    return True
      return False

def isOrList(item):
     if type(item).__name__ =='list':
        if len(item) >2:
            if item[0] =='or':
                return True
        return False
Output 1 :- CLEAR()
 TELL(['implies', 'p', 'q'])
 TELL(['implies', 'r', 's'])
  ASK['implies', ['or', 'p', 'r'], ['or', 'q', 's']])
 : True

Output II   CLEAR()
            TELL('p')
            TELL(['implies', ['and', 'p', 'q'], 'r'])
            TELL(['implies', ['or', 's', 't'], 'q'])
            TELL('t')
             ASK('r')
:True
Output 3 CLEAR()
           TELL('a')
           TELL('b')
           TELL('c')
           TELL('d')
            ASK(['or', 'a', 'b', 'c', 'd'])
```

Output 4.

```
CLEAR()
TELL('a')
TELL(b')
TELL(['or', ['not', 'a'], 'b'])
TELL(['or', 'c', 'd'])
TELL('d')
ASK('c')
False
```

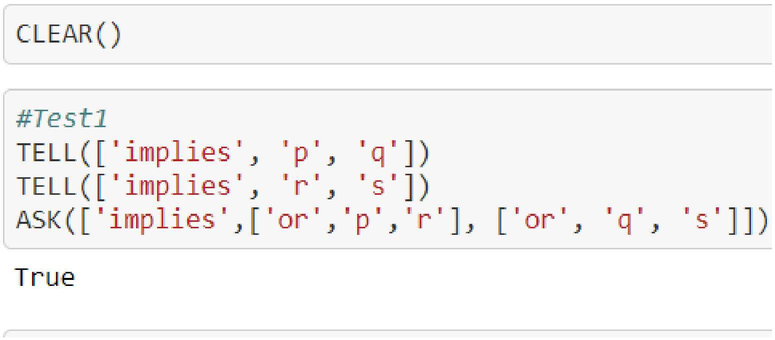
Program:

```
kb = []

def CLEAR():
    global kb
    kb = []

def TELL(sentence):
    global kb

    if isClause(sentence):
        kb.append(sentence)

    else:
        sentenceCNF = convertCNF(sentence)
        if not sentenceCNF:
            print("Illegal input")
            return

        if isAndList(sentenceCNF):
            for s in sentenceCNF[1:]:
                kb.append(s)
        else:
            kb.append(sentenceCNF)

def ASK(sentence):
    global kb

    if isClause(sentence):
        neg = negation(sentence)
```

```
else:
    sentenceCNF = convertCNF(sentence)
    if not sentenceCNF:
        print("Illegal input")
        return
    neg = convertCNF(negation(sentenceCNF))

ask_list = []
if isAndList(neg):
    for n in neg[1:]:
        nCNF = makeCNF(n)
        if type(nCNF).__name__ == 'list':
            ask_list.insert(0, nCNF)
        else:
            ask_list.insert(0, nCNF)
else:
    ask_list = [neg]

clauses = ask_list + kb[:]

while True:
    new_clauses = []
    for c1 in clauses:
        for c2 in clauses:
            if c1 is not c2:
                resolved = resolve(c1, c2)
                if resolved == False:
                    continue
                if resolved == []:
                    return True
                    new_clauses.append(resolved)

        if len(new_clauses) == 0:
            return False

        new_in_clauses = True
        for n in new_clauses:
            if n not in clauses:
                new_in_clauses = False
                clauses.append(n)

        if new_in_clauses:
            return False
    return False

def resolve(arg_one, arg_two):
    resolved = False

    s1 = make_sentence(arg_one)
    s2 = make_sentence(arg_two)

    resolve_s1 = None
    resolve_s2 = None

    for i in s1:
        if isNotList(i):
            a1 = i[1]
            a1_not = True
        else:
            a1 = i
        a1_not = False

    for j in s2:
        if isNotList(j):
            a2 = j[1]
            a2_not = True
        else:
            a2 = j
            a2_not = False

        if a1 == a2:
            if a1_not != a2_not:
                if resolved:
                    return False
                else:
                    resolved = True
                    resolve_s1 = i
                    resolve_s2 = j
                    break
if not resolved:
    return False

s1.remove(resolve_s1)
s2.remove(resolve_s2)

result = clear_duplicate(s1 + s2)

if len(result) == 1:
    return result[0]
elif len(result) > 1:
        result.insert(0, 'or')

    return result

def make_sentence(arg):
    if isLiteral(arg) or isNotList(arg):
        return [arg]
    if isOrList(arg):
        return clear_duplicate(arg[1:])
    return

def clear_duplicate(arg):
    result = []
    for i in range(0, len(arg)):
        if arg[i] not in arg[i+1:]:
            result.append(arg[i])
    return result

def isClause(sentence):
    if isLiteral(sentence):
        return True
    if isNotList(sentence):
        if isLiteral(sentence[1]):
            return True
        else:
            return False
    if isOrList(sentence):
        for i in range(1, len(sentence)):
            if len(sentence[i]) > 2:
                return False
            elif not isClause(sentence[i]):
                return False
        return True
    return False

def isCNF(sentence):
    if isClause(sentence):
        return True
    elif isAndList(sentence):
        for s in sentence[1:]:
            if not isClause(s):
                return False
        return True
    return False

def negation(sentence):
    if isLiteral(sentence):
        return ['not', sentence]
    if isNotList(sentence):
        return sentence[1]

    if isAndList(sentence):
        result = ['or']
        for i in sentence[1:]:
            if isNotList(sentence):
                result.append(i[1])
            else:
                result.append(['not', sentence])
        return result
    if isOrList(sentence):
        result = ['and']
        for i in sentence[:]:
            if isNotList(sentence):
                result.append(i[1])
            else:
                result.append(['not', i])
        return result
    return None

def convertCNF(sentence):
    while not isCNF(sentence):
        if sentence is None:
            return None
        sentence = makeCNF(sentence)
    return sentence

def makeCNF(sentence):
    if isLiteral(sentence):
        return sentence

    if (type(sentence).__name__ == 'list'):
        operand = sentence[0]
        if isNotList(sentence):
            if isLiteral(sentence[1]):
                return sentence
            cnf = makeCNF(sentence[1])
            if cnf[0] == 'not':
                return makeCNF(cnf[1])
            if cnf[0] == 'or':
                result = ['and']
                for i in range(1, len(cnf)):
                    result.append(makeCNF(['not', cnf[i]]))
                return result
            if cnf[0] == 'and':
                result = ['or']
                for i in range(1, len(cnf)):
                    result.append(makeCNF(['not', cnf[i]]))
                return result
            return "False: not"

        if operand == 'implies' and len(sentence) == 3:
            return makeCNF(['or', ['not', makeCNF(sentence[1])], makeCNF(sentence[2])])

        if operand == 'biconditional' and len(sentence) == 3:
            s1 = makeCNF(['implies', sentence[1], sentence[2]])
            s2 = makeCNF(['implies', sentence[2], sentence[1]])
            return makeCNF(['and', s1, s2])

        if isAndList(sentence):
            result = ['and']
            for i in range(1, len(sentence)):
                cnf = makeCNF(sentence[i])

                if isAndList(cnf):
                    for i in range(1, len(cnf)):
                        result.append(makeCNF(cnf[i]))
                    continue
                result.append(makeCNF(cnf))
            return result
```

```
if isOrList(sentence):
    result1 = ['or']
    for i in range(1, len(sentence)):
        cnf = makeCNF(sentence[i])

        if isOrList(cnf):
            for i in range(1, len(cnf)):
                result1.append(makeCNF(cnf[i]))
            continue
        result1.append(makeCNF(cnf))

    while True:
        result2 = ['and']
        and_clause = None
        for r in result1:
            if isAndList(r):
                and_clause = r
                break
        if not and_clause:
            return result1

        result1.remove(and_clause)

        for i in range(1, len(and_clause)):
            temp = ['or', and_clause[i]]
            for o in result1[1:]:
                temp.append(makeCNF(o))
            result2.append(makeCNF(temp))
        result1 = makeCNF(result2)
    return None
    return None

def isLiteral(item):
    if type(item).__name__ == 'str':
        return True
    return False



def isNotList(item):
    if type(item).__name__ == 'list':
        if len(item) == 2:
            if item[0] == 'not':
                return True
    return False



def isAndList(item):
    if type(item).__name__ == 'list':
        if len(item) > 2:
            if item[0] == 'and':
                return True
    return False



def isOrList(item):
    if type(item).__name__ == 'list':
        if len(item) > 2:
            if item[0] == 'or':
                return True
    return False
```

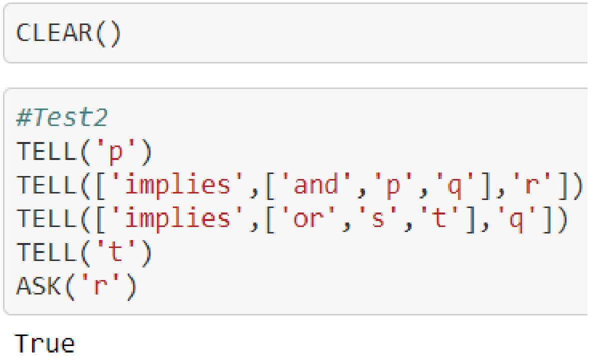
Output:



```
CLEAR()
```

```
#Test1
TELL(['implies', 'p', 'q'])
TELL(['implies', 'r', 's'])
ASK(['implies',['or','p','r'], ['or', 'q', 's']])
```

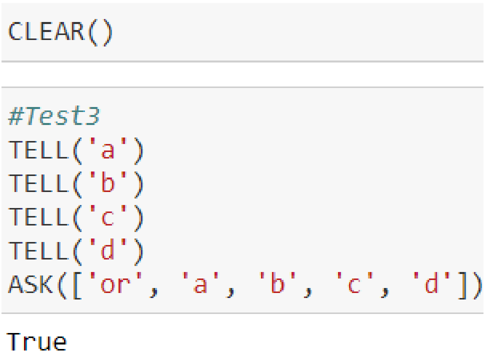
```
True
```



```
CLEAR()
```

```
#Test2
TELL('p')
TELL(['implies',['and','p','q'],'r'])
TELL(['implies',['or','s','t'],'q'])
TELL('t')
ASK('r')
```

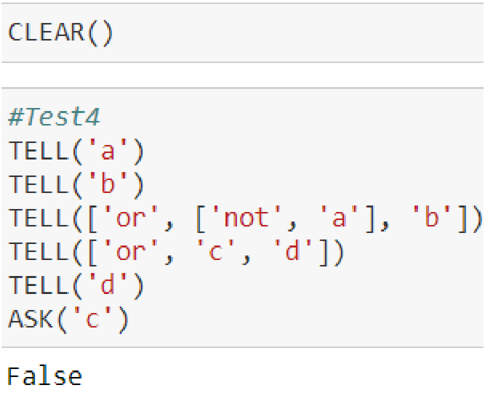
```
True
```



```
CLEAR()
```

```
#Test3
TELL('a')
TELL('b')
TELL('c')
TELL('d')
ASK(['or', 'a', 'b', 'c', 'd'])
```

```
True
```



```
CLEAR()
```

```
#Test4
TELL('a')
TELL('b')
TELL(['or', ['not', 'a'], 'b'])
TELL(['or', 'c', 'd'])
TELL('d')
ASK('c')
```

```
False
```