6.Implementation of the problem-solving strategies: either using Forward Chaining or Backward Chaining.

```
#6:Implementation of the problem solving strategies: using Forward Chaining
from collections import deque
import copy
file=open(input('file:'))
line=file.readlines()
line=list(map(lambda s: s.strip(),line)) #A lambda function can take any number of arguments,
                                       # but can only have one expression.

R = [ ]
for i in range(len(line)):
    k=i+1
    if line[i]=='1) Rules':
        while line[k] != '2) Facts':
            r = deque(line[k].split())
            rhs = r.popleft()
            r.append(rhs)
            R.append(list(r))
            k = k + 1
    elif line[i]=='2) Facts':
        Fact=line[k].split()
    elif line[i]=='3) Goal':
        Goal=line[k]

# ------------------------------
print('PART1. Data')
print(' 1)Rules')
for i in range(len(R)):
    print('     R', i+1, ': ', end=")
    for j in range(len(R[i])-1):
        print(R[i][j], end= ' ')
    print('->', R[i][-1])

print()
print(' 2)Facts')
print('     ', end=")
for i in Fact:
    print(i,' ',end=")
print();print()

print(' 3)Goal')
print('     ', Goal)

# -------------------------------------
```

```
Path=[]
Flag=[]
origin_fact = copy.deepcopy(Fact)

print('PART2. Trace')

# Set initial value
count=0
Yes = False
while Goal not in Fact and Yes==False: #fact When the final element is added to or when it
doesn't work even after finishing it.
    count += 1
    print(' ', end=")
    print('ITERATION',count)
    K=-1
    apply = False
    while K<len(R)-1 and not apply: #until it finds one applicable rule.
        K=K+1
        print('     R', K + 1, ': ', end=")
        for i, v in enumerate(R[K]):     # Print Kth rule (R[K])
            if i < len(R[K]) -1:
                print(v, ", end=")
            else:
                print('->',v, end=")

        if str(K+1) in Flag: #if there is a flag
            b = Flag.index(str(K+1)) +1
            if Flag[b]==[1]:
                print(', skip, because flag1 raised')
            elif Flag[b]==[2]:
                print(', skip, because flag2 raised')

        else: #no flag
            for i, v in enumerate(R[K]):  # Are all the left sides of the kth rule present?
                if i == len(R[K]) -1:
                    continue

                if v in Fact:
                    if R[K][-1] in Fact:  # If the right-hand side already exists
                        print(' not applied, because RHS in facts. Raise flag2')
                        Flag.append(str(K + 1));  Flag.append([2])
                        break
                    elif v == R[K][-2]:
                        apply = True
                        P=K+1
                        break
                else:
                    print(', not applied, because of lacking ', v)
                    break

            if apply:
                Fact.append(R[P-1][-1])
                Flag.append(str(P)); Flag.append([1])
                Path.append(P)
                print(', apply, Raise flag1. Facts ', end='')
                for i in Fact:
                    print(i,' ', end='')
                print()
            elif K== len(R)-1:
                Yes=True
print()
print('PART3. Results')
if Goal in origin_fact:
    print('    ', end='')
    print('Goal A in facts. Empty path.')
else:
    if Goal in Fact:
        print('    ',end='')
        print('1) Goal',Goal,'achieved')
        print('    ', end='')
        print('2) Path:', end='')
        for i in Path:
            print('R', i, ' ', end='')
    else:
        print('1) Goal',Goal,' not achieved')
```

Input & Output

- Input: A text file that contains rules, fact and goal to deal with.

  Example of Input - Testcase 1

  ```
  Test 1. Initial fact in right hand side
  1) Rules
  L A
  K L
  A D
  M D
  Z F B
  F C D
  D A
  ```

2) Facts
A B C
3) Goal
Z

- Output : Data, Trace and Results

Example of Output - Testcase 1

```
forwardChaining.py
  file: test1.txt

  PART1. Data

   1)Rules
      R 1 : A  -> L
      R 2 : L  -> K
      R 3 : D  -> A
      R 4 : D  -> M
      R 5 : F  B  -> Z
      R 6 : C  D  -> F
      R 7 : A  -> D

   2)Facts
      A  B  C

   3)Goal
      Z

  PART2. Trace
   ITERATION 1
      R 1 :A -> L, apply, Raise flag1. Facts  A  B  C  L
   ITERATION 2
      R 1 :A -> L, skip, because flag1 raised
      R 2 :L -> K, apply, Raise flag1. Facts  A  B  C  L  K
   ITERATION 3
      R 1 :A -> L, skip, because flag1 raised
      R 2 :L -> K, skip, because flag1 raised
      R 3 :D -> A, not applied, because of lacking  D
      R 4 :D -> M, not applied, because of lacking  D
      R 5 :F B -> Z, not applied, because of lacking  F
      R 6 :C D -> F, not applied, because of lacking  D
      R 7 :A -> D, apply, Raise flag1. Facts  A  B  C  L  K  D
   ITERATION 4
      R 1 :A -> L, skip, because flag1 raised
```

```
    R 2 :L -> K, skip, because flag1 raised
    R 3 :D -> A not applied, because RHS in facts. Raise flag2
    R 4 :D -> M, apply, Raise flag1. Facts  A  B  C  L  K  D  M
 ITERATION 5
    R 1 :A -> L, skip, because flag1 raised
    R 2 :L -> K, skip, because flag1 raised
    R 3 :D -> A, skip, because flag2 raised
    R 4 :D -> M, skip, because flag1 raised
    R 5 :F B -> Z, not applied, because of lacking  F
    R 6 :C D -> F, apply, Raise flag1. Facts  A  B  C  L  K  D  M  F
 ITERATION 6
    R 1 :A -> L, skip, because flag1 raised
    R 2 :L -> K, skip, because flag1 raised
    R 3 :D -> A, skip, because flag2 raised
    R 4 :D -> M, skip, because flag1 raised
    R 5 :F B -> Z, apply, Raise flag1. Facts  A  B  C  L  K  D  M  F  Z

PART3. Results
   1) Goal Z achieved
   2) Path:R 1  R 2  R 7  R 4  R 6  R 5
Process finished with exit code 0
```