SUB: AI Lab Exam.

NAME: PRATHIBHA.R
CLASS: 5B
USN: 1BM18CS074

```
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 = convert-CNF(sentence)
        if not sentenceCNF:
            print("Illegal input").
```

```
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
```

Date _____

```
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 printKB()
    global kb
    for k in kb
        Print(k)

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 is Not List(1):
        a1 = i[1]
        a1_not = True.
    else:
        a1 = i:
        a1_not = False.
```

```
for f in s2:
    if isNotList(f):
        a2 = f[1].
        a2_not = True.

    else:
        a2 = f
        a2_not = False

    if a1 == a2:
        if a1_not != a2_not:
            if resolved:
                return False.
            else
                resolved = True
                resolve_s1 = e.
                resolve_s2 = f.
                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 p in range(0, len(arg)):
        if arg[p] not in arg[p+1:]:
            result.append(arg[p])
    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[1]):
            return 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]]))

    if operand == 'implies' and len(sentence) == 3:
        return makeCNF(['or', ['not', makeCNF(sentence[1])],
                        makeCNF(sentence[2])])
    if operand == 'bidirectional' and len(sentence) == 3:
        s1 = makeCNF(['implies', sentence[1], sentence[2]])
        s2 = makeCNF(['implies', sentence[2], sentence[1]])
        return makeCNF(['and', s1, s2])
```

Date .....

Expt. No. .....

Page No. .....

Other funcs

Prathibha
1BM18CS076

Test case 1.

```
TELL ( ['implies', 'a', 'b'])
TELL ( 'implies', 'c', 'd')
ans = ASK ( ['implies', ['or', a, 'c'], ['or', 'b', 'd']])
if ans == True:
    print("True")
else:
    print ("False")
```

Testcase 2

```
TELL ('a')
TELL('b')
TELL ('or', ['~~not~~', 'a'], ['b'])
TELL ('or', 'c', 'd'])
TELL('d')
ans1 = ASK = ('c')
if ans1 == True:
    print (True)
else
    print ("False")
```

Testcase 3

```
TELL ('a')
TELL('b')
TELL('c')
TELL('d')
ans2 = ASK (['or', 'a', 'b',
    'c', 'd'])

if ans2 == True.
    print ("True")
else
    print (False)
```

Teacher's Signature : _____