CSE 4301/5290 Homework 4
Due: Nov 12, Wed, 5pm; Submit Server: class = ai , assignment = hw4

For programming problems (LISP/Java/C/C++/Python):

- Submit:
  - all files that are needed to compile and run
  - README.txt with compilation & run instructions
- Your program should compile and run on `code.fit.edu` (Linux, remote access via ssh) or `hopper.cs.fit.edu` (Windows, remote access via Remote Desktop).

1. Q7.10, p281, 3Ed (Q7.8, p237, 2Ed). For 3Ed, add part g: $(Big \wedge Dumb) \vee \neg Dumb$

2. In proof by contradiction (using the resolution inference rule), when $KB \wedge \neg\alpha$ is *unsatisfiable*, we know α is true. What do we know about α when $KB \wedge \neg\alpha$ is *satisfiable*? When can we know that α is false? Explain your answers.

3. Using a truth table, prove that:
   (a) $(a \vee b) \wedge (\neg b \vee c)$ entails $a \vee c$ [correct "resolution"]
   (b) $(a \vee b \vee c) \wedge (\neg b \vee \neg c \vee d)$ does not entail $a \vee d$. [incorrect "resolution"]

4. Q7.2, p280, 3Ed (Q7.9, p238, 2Ed): Write sentences in propositional logic, translate them into clauses, use resolution to infer answers for the three queries.

5. Programming: Given clauses (CNF) in propositional logic, use resolution with at least 3 strategies to gain speed [2 discussed in class plus an additional one—described in the comments] to solve:

   (a) Wumpus, p247, 3Ed (p208, 2Ed): The initial KB has $R_1 - R_3$; percepts are R_4 and R_5; queries are:
      i. a pit at [1,2]?
      ii. a pit at [2,2]?

   (b) Unicorn, Q7.2, p280, 3Ed (Q7.9, p238, 2Ed): no percepts, three queries.

   Represent a clause (disjunction) using a string or a list. For example, $a \vee \neg b \vee c$ is represented as:

```
"a !b c"
(a (not b) c)
```

   Represent CNF using a string or a list. For example, $(a \vee \neg b \vee c) \wedge (\neg a \vee d)$ is represented as:

```
"(a !b c) (!a d)"
((a (not b) c) ((not a) d))
```

   For c/c++/java/python, you have at least three modules: KB, TestWumpus, and TestUnicorn. Functions in your implementation (stated in LISP) include:

```
; add percepts (a list of clauses) to kb and return the updated kb
(defun tell-kb (kb percepts)  ...)

; given kb (a list of clauses), use resolution to infer an answer
;      for the query
; return answer for the query
(defun ask-kb (kb query)  ...)

; initialize kb, add percepts to kb,
;   print queries and corresponding answers
; return 'done
(defun test-wumpus ()
  (let* ((kb ...) ...)
   ...
  )
)

(defun test-unicorn ()
  (let* ((kb ...) ...)
   ...
  )
)
```

CSE 5290 only

6. Formulate proof by contradiction using the resolution inference rule into a state-space search problem that finds the shortest proof (fewest applications of the resolution inference rule). For using A*, discuss a (non-constant-zero and non-constant-one) heuristic and explain why it is *admissible*.

7. Programming: Given logical sentences, convert them into CNF in the format used in the programming Problem 5 above. The allowed connectives are:

Connective	prefix	infix
$\wedge$	and	&
$\vee$	or	|
$\neg$	not	!
$\Rightarrow$	imply	=>
$\Leftrightarrow$	bicond	<=>

   For example, $a \wedge b \Rightarrow c$ is represented as:

```
"(a & b) => c"
(imply (and a b) c)
```

   For c/c++/java/python, you have at least four modules: ConvertToCNF, TestToyConvert, TestWumpusConvert, and TestUnicornConvert. The functions/methods (stated in LISP) include:

```
; convert sentence into CNF and return CNF
(defun convert-to-cnf (sentence) ...)

; convert toy kb to CNF, return CNF
; print each sentence and its cnf
(defun test-toy-convert ()
  (let* ((kb '(
           (and (not a) b)     ; "!a & b"
           (or b (and c d))    ; "b | (c & d)"
           (not (or d e))      ; "!(d | e)"
           (not (and e f))     ; "!(e & f)"
           (imply (and f g) h) ;  "(f & g) => h"
           (bicond (and h (not i)) (and j k)) ; "(h & !i) <=> (j & k)"
       )))
    ...)
  ...
)

; convert the wumpus initial kb (Problem 5a) to CNF, return CNF
; print each sentence and its CNF
(defun test-wumpus-convert () ...)

; convert the unicorn intital kb (Problem 5b) to CNF, return CNF
; print each sentence and its CNF
(defun test-unicorn-convert () ...)
```