Homework 1
CS161, Spring 2014
Due: Monday, April 14 by midnight. Submit via CourseWeb.

For this homework you are to implement 5 Lisp functions that will operate on Lisp data structures called *frames*. **The example frames on which you will test your functions appear in the file: MerryWidowMurderer-SentRepres**. By examining these examples of instantiated frames you will see that each *frame* conforms to the following syntax:

frame → (*pred {slot filler}*)* | ( )   where {}* indicates zero or more of whatever is inside the curly brackets and | indicates an alternative.
pred → *atom*
slot → *atom*
filler → *frame* | *gap*
gap → *atom* | *variable*
variable → *(V atom)*

To improve readability, we will often indent slot-filler pairs and highlight *slot* names with **bold** while giving *gaps* a numeric suffix beginning with 0 (e.g. AG0, OBJ0). **In this homework we will ignore variables.**

Note: The order of the slots should not matter to the functions you will define. For example, your functions should work whether or not Charlotte Newton is represented as:

```
(HUMAN F-NAME (CHARLOTTE)
       L-NAME (NEWTON)
       GENDER (FEMALE))
```

or

```
(HUMAN GENDER (FEMALE)
       L-NAME (NEWTON)
       F-NAME (CHARLOTTE))
```

Note: A *gap* is a Lisp atom that can have other atom (or a frame) as it value. For example, instead of:

```
(HUMAN F-NAME (CHARLOTTE)
       L-NAME (NEWTON))
```

We could have:

```
(HUMAN F-NAME (CHARLOTTE)
       L-NAME LNM001)
```

where `LNM001 = (NEWTON)`

See the last example in file MerryWidowMurderer-SentRepres for a frame with gaps in it.

The following lists and describes the functions that you must implement for HW1:

Problem 1: (FILLER slot frame)

Takes a **slot** and a **frame** and returns the *filler* of that slot. Note that **slot** refers to a top-level slot (not one embedded within a subframe). If there is not a top-level **slot** in the **frame**, then NIL is returned.

Examples:

`(FILLER ‘AGENT CON-SENT1)` returns:

```
(HUMAN F-NAME (CHARLOTTE)
       L-NAME (NEWTON)
       GENDER (FEMALE)
       AGE (RANGE FROM (13)
                  TO (19)
                  UNIT (YEAR)))
```

`(FILLER ‘TIME CON-SENT2)` returns: `(FUTURE)`
`(FILLER ‘TIME CON-SENT1)` returns: `NIL`
`(FILLER ‘OBJECT SENT3-GAPPED)` returns: `OBJ001`

Function Skeleton:

```
(defun FILLER (slot frame)
  ; Your code here
)
```

Problem 2: (PATH-SL slots concept)

Takes a list of slots and uses them to path into a concept (i.e. an instantiated frame) and returns the frame that is being sought. Notice that the preds in the frames are ignored.

Examples:

```
(PATH-SL '(OBJECT OBJECT F-NAME) CON-SENT9)
returns:  (SEMANTHA)

(PATH-SL '(EXP-VIOL OBJECT GENDER) CON-SENT3)
returns:  (MALE)

(PATH-SL '(FAMREL OBJECT) CON-SENT2)
returns:  (HUMAN F-NAME (CHARLOTTE)
                 L-NAME (NEWTON)
                 GENDER (FEMALE))

(PATH-SL '(EXP-VIOL AGENT F-NAME) SENT3-GAPPED)
returns:  (CHARLOTTE)

(PATH-SL '(EXP-VIOL AGENT) SENT3-GAPPED)
returns:  AG002
```

Function Skeleton:

```
(defun PATH-SL (slots concept)
  ; Your code here
)
```

Problem 3: (UNGAP atom)

Takes an atom that has a atom or instantiated frame as its value. UNGAP generates a new frame with <u>all</u> gaps removed. This is done by replacing each gap, recursively, with it whatever frame is its value.

<u>Examples:</u>

`(UNGAP ‘SENT3-GAP)` returns a frame that has the same structure as the value of CON-SENT3.

```
(UNGAP ‘EXPV001)
```
returns:
```
(SEE AGENT (HUMAN F-NAME (CHARLOTTE)
                  L-NAME ( )
                  GENDER (FEMALE))
     OBJECT (HUMAN F-NAME (CHARLES)
                   L-NAME ( )
                   GENDER (MALE)))
```

If a frame has no gaps in it, then UNGAP returns a copy of that frame.

```
(UNGAP CON-SENT1)
```
returns:
```
(STATE TYPE (EMOTION SENTIM (POS)
                     SCALE (>NORM))
       AGENT (HUMAN F-NAME (CHARLOTTE)
                    L-NAME (NEWTON)
                    GENDER (FEMALE)
                    AGE (RANGE FROM (13)
                               TO (19)
                               UNIT (YEAR))))
```

<u>Function Skeleton:</u>

```
(defun UNGAP (atom)
  ; Your code here
)
```

Problem 4: (ADD-SF slot filler frame)

Adds a top-level **slot** with **filler** to a **frame**. If this **slot** already appears at the top level of the **frame**, then ADD-SF replaces the **slot** with the new **filler**.

Examples:
```
(ADD-SF 'L-NAME '(OAKLEY) AG001)
```
returns:
```
(HUMAN F-NAME (CHARLES)
       L-NAME (OAKLEY)
       GENDER (MALE))
```

```
(ADD-SF 'TIME '(PAST) 'CON004)
```
returns:
```
(SEE AGENT AG002
     OBJECT OBJ002
     TIME (PAST))
```

Function Skeleton:

```
(defun ADD-SF (slot filler frame)
  ; Your code here
)
```

Problem 5: (SAME-SF frame1 frame2)

Returns T if frame1 and frame2 have the same slot-filler structure.

Example:

```
(SAME-SF
  (UNGAP CON004)
  ‘(SEE OBJECT (HUMAN F-NAME (CHARLES)
                      L-NAME ( )
                      GENDER (MALE))
        AGENT (HUMAN GENDER (FEMALE)
                     F-NAME (CHARLOTTE)
                     L-NAME ( )))
```

Returns: T
(because they have the same slot-filler structure. Remember, the order in which the slots appear at a given level is irrelevant.)

Function Skeleton:

```
(defun SAME-SF (frame1 frame2)
  ; Your code here
)
```