Homework 2
CS161, Spring 2014
Due: Monday, April 28 by midnight. Submit via CourseWeb.

In this homework some of the Lisp functions that you define will manipulate 4 global variables: LEXMEM, WKMEM, DEMEM, and USEDMEM (each with NIL as its initial value).

LEXMEM (Lexical Memory)

LEXMEM will store words/phrases, along with their associated frames and demons. Here is LEXMEM's structure:

LEXMEM = (((*phrase*) *frame* (*dem-inst**))
((*phrase*) *frame* (*dem-inst**))
…
((*phrase*) *frame* (*dem-inst**)))

where *phrase* → word+
where *dem-inst* → (demon-name arguments*)

(remember, * = 0 or more, + = 1 or more, and +2 = 2 or more)

So a *phrase* consists of a list of one or more words. A *dem-inst* is a list with a demon name, followed by zero or more arguments.

WKMEM (Working Memory)

WKMEM has the following structure:
WKMEM = (CON1 CON2 CON3 …) where the value of each CONcept atom will be an instantiated frame. Atoms in this list must begin with the prefix 'CON.

DEMEM (Demon Memory)

DEMEM is a list with zero of more demon instances:
DEMEM = (*dem-inst**)

By convention, we will prefix each demon name with DM-. Here is an example of an *instance* of the demon DM-EXP:
(DM-EXP CON3 AGENT BEF HUMAN)
You will define this demon in this homework (see Problem #8 below).

USEDMEM (Used Memory)

USEDMEM = (CON3 CON1 …) will hold CON atoms that have been used. Whenever a frame FR1 uses another frame FR2 to fill a gap in FR1, then the CON atom of FR2 will be added to USEDMEM. Atoms in this list must begin with the prefix 'CON.

You will write the following Lisp functions, some of which will access these global variables.

Problem 1: (ADD-LEX phrase frame demons)
Adds a **phrase**, **frame** and **demons** to LEXMEM.

Repeated executions of ADD-LEX will grow LEXMEM (it's returned value is unimportant). If you add the same phrase twice, the second instance of ADD-LEX will replace the same **phrase** with the new **frame** and **demons**. For example, if you first do:

(ADD-LEX '(HOT DOG) '(FOOD) NIL)

...and then later do:

(ADD-LEX '(HOT DOG) '(FOOD **TYPE** (JUNK)) '((DM-HEART-ATTACK)))

...then there will still be only one (HOT DOG) phrase in LEXMEM, with the associated new frame (FOOD **TYPE** (JUNK)) and associated demon list ((DM-HEART-ATTACK)).

Example 1: (using words from sentence 1of the story)
(ADD-LEX
'(TEENAGER)
'(MODIF **AGE** (RANGE **FROM** (13) **TO** (19) **UNIT** (YEAR)))
'((DM-MODIF HUMAN AFTER)))

After executing ADD-LEX, global variable LEXMEM =
(((TEENAGER)
(MODIF **AGE** (RANGE **FROM** (13) **TO** (19) **UNIT** (YEAR)))
((DM-MODIF HUMAN AFTER))))

Example 2:
(ADD-LEX
'(CHARLOTTE)
' (HUMAN **F-NAME** (CHARLOTTE)
L-NAME LNM
GENDER (FEMALE))
'((DM-LNAME L-NAME)))

After executing ADD-LEX
LEXMEM =
(((CHARLOTTE)
(HUMAN **F-NAME** (CHARLOTTE)
L-NAME LNM
GENDER (FEMALE))
((DM-LNAME L-NAME)))
((TEENAGER)
(MODIF **AGE** (RANGE **FROM** (13) **TO** (19) **UNIT** (YEAR)))
((DM-MODIF HUMAN AFTER))))

Example 3:
(ADD-LEX
'(TEENAGER)
'(MODIF **AGE** (RANGE **FROM** (13) **TO** (19) **UNIT** (YEAR)))
'((DM-MODIF HUMAN AFTER)))
After executing this ADD-LEX, LEXMEM looks the same.

Example 4:
(ADD-LEX
'(NEWTON)
'(UNKNOWN **WORD** (NEWTON))
' ( ))

After executing ADD-LEX,
LEXMEM =
(((NEWTON)
(UNKNOWN **WORD** (NEWTON))
NIL)
((CHARLOTTE)
(HUMAN **F-NAME** (CHARLOTTE)
L-NAME LNM
GENDER (FEMALE))
((DM-LNAME L-NAME)))
((TEENAGER)
(MODIF **AGE** (RANGE **FROM** (13) **TO** (19) **UNIT** (YEAR)))
((DM-MODIF HUMAN AFTER))))

Here are the frames and demons associated with the remaining words in sent 1. Use ADD-LEX to grow LEXMEM with these phrases.

...we notice that the word FEELS is ambiguous. In the Merry Widow Murderer story, in SENT-1it results in an emotional STATE, while in SENT-9 it results in a BELIEVE. We will handle ambiguity later, so for now, the frame for the ambiguous phrase "feels" will be empty.

phrase: (FEELS)
frame: NIL
demons: ((DM-DISAMB-FEELS))

phrase: (EXCITED)
frame: (EMOTION **SENTIM** (POS) **SCALE** (>NORM))
demons: NIL

phrase: (PERIOD)
frame: (PERIOD)
demons: NIL

Problem 2: (NEXT-PH WRDLST LEXIC)

Takes a list of words **WRDLST** and a lexicon **LEXIC** (with the same structure as LEXMEM) and returns a list with structure:

((*phrase frame demons**) *rest-of-sentence*)

...where, if a match is found in the lexicon:

- *Phrase* is a list of the words that appear at the very front of **WRDLST** and that matches (exactly) the longest phrase found in **LEXIC**
- *Frame* is the associated frame matched in the lexicon
- *Demons** is a list of zero or more demons associated with *phrase* in the lexicon
- *Rest-of-sentence* is the rest of **WRDLST** with *phrase* removed.

If a match is not found in the lexicon, then that means we ran through all of our **LEXIC** triplets and couldn't find a phrase that matched whatever word was at the start of our **WRDLST**. In other words, if the **LEXIC** argument to NEXT-PH is NIL, then no word will be found in this empty lexicon, and for every word not found in **LEXIC**, NEXT-PH will return a phrase-frame-demon triplet consisting of:

(((*word*) (UNKNOWN **WORD** (*word*)) ( )) *rest-of-sentence*)
phrase *frame* *demons*

...where:

- *Word* is the word from the sentence that could not be matched in the lexicon.
- (UNKNOWN **WORD** (*word*)) is the frame that goes in the frame (second) position of the return.
- And the demon position (third) is NIL. (note: in the case of a word not appearing in the lexicon, the demon position will always be NIL because we have no matched demons from a lexicon entry to populate this position!)

Let us refer to the lexicon (that ADD-LEX creates after processing the 6 words/phrases in sentence 1 of the story) as LEXIC6.

Example 1:
MWM-SENT1 = '(TEENAGER CHARLOTTE NEWTON FEELS EXCITED PERIOD)

(NEXT-PH MWM-SENT1 LEXIC6)
returns:
(((TEENAGER)
(MODIF **AGE** (RANGE **FROM** (13) **TO** (19) **UNIT** (YEAR)))
((DM-MODIF HUMAN AFTER))) CHARLOTTE NEWTON FEELS EXCITED PERIOD)

(more examples on the next page)

Example 2:
FRAGM-SENT2 = '(MOVE IN WITH HER MOTHER)

LEXIC-TEST =
'(((NEWTON) (UNKNOWN **WORD** (NEWTON)) NIL)
  ((MOVE) (MOVE **BODYPART** BPRT) NIL)
  ((CHARLOTTE) (HUMAN **F-NAME** (CHARLOTTE)) NIL)
  ((MOVE IN WITH) (CO-HABITATE **AGENT** AG) ((DM-EXP)))
  ((WITH) (WITH) NIL)))

(NEXT-PH FRAGM-SENT2 LEXIC-TEST)
returns:
(((MOVE IN WITH) (CO-HABITATE **AGENT** AG) ((DM-EXP))) HER MOTHER)

Example 3:
(NEXT-PH '(NEWTON FEELS EXCITED) NIL)
returns:
(((NEWTON) (UNKNOWN **WORD** (NEWTON)) ( )) FEELS EXCITED)

Example 4:
FRAGM-SENT3 = '(MY KINGDOM FOR A MATCH)

(NEXT-PH FRAGM-SENT3 LEXIC-TEST)
returns:
(((MY) (UNKNOWN WORD (MY)) ( )) KINGDOM FOR A MATCH)

(NEXT-PH '(KINGDOM FOR A MATCH) NIL)
returns:
(((KINGDOM) (UNKNOWN **WORD** (KINGDOM)) ( )) FOR A MATCH)

(NEXT-PH '(FOR A MATCH) NIL)
returns:
(((FOR) (UNKNOWN **WORD** (FOR)) ( )) A MATCH)

Problem 3: (NEWATM ATM)

This function takes an atom ATM and creates a new, unique atom out of it. Each call to NEWATM creates a new, unique instance of ATM. Note, this function is simply used to obtain a new, unique name for our input atom, it will not perform any binding. We will create a unique atom name by using **ATM** as the prefix of an increasing number. To do so, we'll use the **gensym** function with its counter starting at 1 (see docs and discussion slides for using gensym). NB., your generated symbols need not necessarily begin with 1 as the numeric suffix (some systems return 3001 for the start) so long as the returned symbols are still unique (which gensym will handle)!

Examples:

(NEWATM 'AG) returns: AG1
(NEWATM 'CON) returns: CON2
(NEWATM 'AG) returns: AG3
(NEWATM 'CON) returns: CON4
(NEWATM 'OBJ) returns: OBJ5
...and so on.

NEWATM will be used to create new CONcept atoms in WKMEM and also to create new gaps in frame instances.

This function is very simple and can be accomplished in 1 line, but requires familiarity with a couple of extra functions:

- **(gensym *string*)**
  Takes a string input and returns a new, unique, unbound symbol that is equal to the input string + a number at the end. This number at the end is equal to the global variable: *gensym-count*, which gensym then increments by 1 before returning.

  Example:
  ; Start our count at 1
  (setq *gensym-count* 1)

  ; First call, *gensym-count* is equal to 1
  (gensym "test")
  returns:
  TEST1
  ; Second call, *gensym-count* is equal to 2
  (gensym "test")
  returns:
  TEST2

- **(string *symName*)**
  Takes a symbol input and returns a string representation of that string.

See how to solve NEWATM now? :)

Problem 4: (UNIQUE-GAPS FRAME)

This function uses NEWATM to replaces every *gap* in **FRAME** with a unique gap atom.

Examples:

(UNIQUE-GAPS '(PRED1 **SLOTA** SLA
SLOTB (PRED2 **SLOTB** SLB)))

returns:
(PRED1 **SLOTA** SLA1
SLOTB (PRED2 **SLOTB** SLB2))

...if you then later called:
(UNIQUE-GAPS '(PRED1 **SLOTA** SLA
SLOTB (PRED2 **SLOTB** SLB
SLOTA SLA)
SLOTC (PRED2 **SLOTD** SLD)))

returns:
(PRED1 **SLOTA** SLA3
SLOTB (PRED2 **SLOTB** SLB4
SLOTA SLA5)
SLOTC (PRED2 **SLOTD** SLD6)))

Problem 5: (INSTAN-CON FRAME WKM)

This function instantiates a **FRAME** by making any *gaps* in **FRAME** unique (using UNIQUE-GAPS). It creates a new unique CON atom (using NEWATM with symbol prefix 'CON) with the instantiated frame as its value and adds that CON atom to the end of working memory **WKM**. It returns the new value of WKM and, as a side-effect, it sets the global variable WKMEM = WKM.

Examples: (assume WKMEM1 had 3 atoms already that were named using NEWATM)
WKMEM1 = (CON1 CON2 CON3)
(INSTAN-CON '(HUMAN **F-NAME** (CHARLOTTE)
L-NAME LNM
GENDER (FEMALE))
WKMEM1)

returns: (and sets WKMEM to)
(CON1 CON2 CON3 CON4)

where:
CON4 = (HUMAN **F-NAME** (CHARLOTTE)
L-NAME LNM5
GENDER (FEMALE))

Notice that LNM has been "uniquified" to LNM5.

Problem 6: (SPAWN PARTIAL-DEMS MYCON)

This function takes a list of partial demon instances **PARTIAL-DEMS** and completes each partial demon instance by adding the atom **MYCON** as a first argument to each instance. SPAWN returns a completed version of PARTIAL-DEMS. As a side-effect, SPAWN adds these completed demon-instances to the front of the global variable **DEMEM**.

Examples: (assume that the global variable DEMEM has one dem-inst, (DM-FAUX))

```
DEMEM = ((DM-FAUX))
(SPAWN   '( (DM-EXP HUMAN BEF AGENT)
              (DM-EXP EMOTION AFT TYPE) )
           'CON99)
```

returns:

```
( (DM-EXP CON99 EMOTION AFT TYPE)
  (DM-EXP CON99 HUMAN BEF AGENT) )
```

and as a side-effect:

```
DEMEM =
( (DM-EXP CON99 EMOTION AFT TYPE)
  (DM-EXP CON99 HUMAN BEF AGENT)
  (DM-FAUX) )
```

Problem 7: (SRCH ATMLST MYATM DIR PRED)

This is a utility function that is used by some demons. **SRCH** searches a list of atoms **ATMLST**, starting with atom **MYATM** and searching in direction **DIR**. **SRCH** looks for an atom whose value is a frame with top predicate **PRED** and returns that atom if found (or returns NIL).

DIR can have one of the following 4 directions:

- *AFT* -- closest after (to the right of) **ATM**
- IM-AFT -- immediately after **ATM**
- BEF -- closest before (to the left of) **ATM**
- IM-BEF -- immediately before **ATM**

Examples:
ATMLST1 = (CON1 CON2 CON3 CON4 CON5 CON6)
where:

CON1 = (MODIF **AGE** AGE7)
CON2 = (HUMAN **F-NAME** (CHARLOTTE))
CON3 = (UNKNOWN **WORD** (NEWTON))
CON4 = (AMBIG)
CON5 = (EMOTION **SENTIM** (POS) **SCALE** (>NORM))
CON6 = (PERIOD)

(SRCH ATMLST1 'CON1 'AFT 'EMOTION) returns CON5
(SRCH ATMLST1 'CON1 'IM-AFT 'EMOTION) returns NIL
(SRCH ATMLST1 'CON2 'IM-AFT 'UNKNOWN) returns CON3
(SRCH ATMLST1 'CON2 'IM-BEF 'MODIF) returns CON1
(SRCH ATMLST1 'CON5 'BEF 'HUMAN) returns CON2

Problem 8: (BIND GAP FOUND)

This function expects **FOUND** to be a CON atom and **GAP** to be the gap to some slot. **BIND** sets **GAP = FOUND** and, as a side effect, adds **FOUND** to the USEDMEM global.

Example:

USEDMEM = NIL
CON1 = (HUMAN **F-NAME** (CHARLOTTE))

(BIND 'NEW-NAME 'CON1)

returns:
CON1
(and as side effects):

USEDMEM = (CON1)
NEW-NAME = 'CON1

Demons

Let us now discuss how demons work. Demons are defined just like any other function. However, unlike standard functions, many different instances of the same demon can be active. A demon instance can remain active and search some environment at different times, until some condition is satisfied. If the condition is satisfied the demon instance will "fire" and do some action.

Actions that demon instances commonly perform include:

- Bind a gap in a frame with another concept.
- Insert a slot-filler pair into a frame.
- Disambiguate an ambiguous structure and/or spawn other demons.
- Request to die.
- Create episodic structures (that are stored in episodic memories).

In this homework you will define and try out one demon: DM-EXP.

Problem 9: (DM-EXP MYCON PRED DIR MYSLOT)

DM-EXP looks for a CON atom in the global variable WKMEM with top predicate **PRED**. If it finds this PRED, then let's call that CON atom *found*. If it finds *found* then this DM-EXP instance binds *found* to the top-level gap associated with **MYSLOT** in **MYCON** (using the BIND function you developed in problem #8). It returns (DIE) and (as a side-effect of having used the BIND function from problem #8) now *found* has been added to the global variable USEDMEM. When looking for *found*, DM-EXP starts searching at **MYCON** and looks in the global variable WKMEM in direction **DIR**. If DM-EXP is unsuccessful it returns NIL.

Examples:

Consider this instance of the demon **DM-EXP**:
(DM-EXP CON3 HUMAN BEF AGENT)

This demon "works for" the frame associated with CON3 in WKMEM. It is looking for a HUMAN appearing BEFore CON3. (If there is more than one HUMAN in WKMEM, it will find the closest one to its left.) DM-EXP will use the utility function **SRCH** to do this search. When it has found something (i.e., *found*), it will do the following:

1. It will BIND *found* to the gap associated with the AGENT slot of CON3 (and therefore add *found* to the global variable USEDMEM from calling BIND). That is, this gap will get *found* as its value.
2. Since it has found something, it will return (DIE).

We will implement demons via polling (i.e. repeatedly executing demons until they are successful or until they decide they can die).

Problem 10: (POLL-DEMS DEMLST)

POLL-DEMS goes through a list of demon instances in **DEMLST** and polls them (i.e. executes each one). When a demon is successful, it will return (DIE) to **POLL-DEMS**, which will cause **POLL-DEMS** to remove that demon instance from **DEMLST**. If any demon returns (DIE) then POLL-DEMS will re-poll the remaining demons. It does this until no demon returns (DIE). That is, POLL-DEMS polls demons repeatedly until they are all quiescent (they all return NIL). **POLL-DEMS** returns DEMLST.

Here is a simplified example:
Assume that the sentence is: "Charlotte eats"

USEDMEM = NIL
WKMEM = '(CON1 CON2)

where:
CON1 = (HUMAN **F-NAME** (CHARLOTTE))
CON2 = (EAT **AGENT** AG4 **OBJECT** OBJ5)
CON3 = (FOOD **TYPE** (BACON))
DEMLST1 = ((DM-EXP CON2 FOOD AFT OBJECT)
(DM-EXP CON2 HUMAN BEF AGENT))

(POLL-DEMS DEMLST1) will first execute demon instance:
(DM-EXP CON2 FOOD AFT OBJECT)

This demon instance will not find what it is looking for, so it will return NIL to POLL-DEMS. Thus, it remains in DEMLST1.

POLL-DEMS will then execute demon instance:
(DM-EXP CON2 HUMAN BEF AGENT).

This one will be successful and cause the following to occur:

- AG4 = CON1
  Note: AG4 is the gap for the AGENT slot in the EAT frame associated with CON atom CON2. This instance of DEM-EXP works for CON2 and is looking for a HUMAN before CON2 and it wants to fill the **AGENT** slot with what it finds. When it finds CON1 it will use PATH-SL (from HW-1) to get the associated gap, which in this case, is AG4. It then calls the function BIND to set AG4 = CON1.
- BIND will add CON1 to USEDMEM, so now USEDMEM = (CON1)
- This instance of DM-EXP returns: (DIE)

Since this demon instance returned (DIE), **POLL-DEMS** removes it from DEMLST1. If any demon instance was removed (returned (DIE)), **POLL-DEMS** re-executes all demon instances remaining in DEMLST1. In this case there is only one demon instance remaining, and it will fail and return NIL. Since all demons in the demon list returned NIL on this run through, **POLL-DEMS** terminates and returns DEMLST1. As a side-effect, the global variable:
DEMEM = DEMLST1

Now, let's add "bacon", so we have just read "Charlotte eats bacon".
At this point we will have the (FOOD …) frame (CON3) added to WKM1:

WKMEM = '(CON1 CON2 CON3)

and our one remaining demon instance:
DMLST1 = ((DM-EXP CON2 FOOD AFT OBJECT))

If we execute POLL-DEMS now, then this instance should now be successful, which will cause the following to happen:

- This instance of DEM-EXP will find CON3 and call BIND to set OBJ5 = CON3, and add CON3 to USEDMEM, so USEDMEM = (CON3 CON1)
- This demon instance returns (DIE) and so **POLL-DEMS** removes it from DMLST1
- Since DMLST1 is now empty, POLL-DEMS terminates and returns: NIL

As a side-effect, POLL-DEMS sets the global variable:

DEMEM = DMLST1 = ( )

Problem 11: (TOP-CON WKM USED)

Here is one more utility function to define (easiest utility saved for last :-). **WKM** and **USED** are both lists of atoms. This function goes through the atoms in **WKM** and returns a list of all atoms that do NOT appear in **USED**.

Example:
(TOP-CON '(CON1 CON2 CON3) '(CON3 CON1))
returns:
(CON2)

Problem 12: (C-ANALYZER SENT LEXIC)

This function takes a sequence of words **SENT** and a lexicon **LEXIC** as input. It repeatedly does the following:

1. Use NEXT-PH to get the next longest phrase at the front of **SENT** and remove it from **SENT**.
2. Use INSTAN-CON to instantiate the associated frame (i.e., the frame associated with the phrase that NEXT-PH found).
3. Use SPAWN to spawn the associated demon instances.
4. Use POLL-DEMS to repeatedly poll the demon instances until the remaining ones in DEMEM are all quiescent.

Repeat the above until SENT = NIL

C-ANALYZER then uses the UNGAP (from HW-1) and TOP-CON to return a list containing any TOP (i.e. unused) frame from the CON atoms in WKMEM. If every demon does its job correctly, then this list should contain only one UNGAP-ed frame.

That is, the return value is the result of calling UNGAP on each CON atom in:
(TOP-CON WKMEM USEDMEM). You'll probably want to define a helper function to do this!

Example:

```
TEST-SENT = '(CHARLOTTE EATS HOT DOGS)
TEST-LEXMEM =
        (((HOT DOGS)
                (FOOD TYPE (WEENER) AMNT (MULTI))
                ( ))
            ((EATS)
                (EAT AGENT AG1 OBJECT OBJ2)
                ((DM-EXP HUMAN BEF AGENT)
                 (DM-EXP FOOD AFT OBJECT)))
            ((CHARLOTTE)
                (HUMAN F-NAME (CHARLOTTE))
                ( )))

(C-ANALYZER TEST-SENT   TEST-LEXMEM)
returns:
((EAT AGENT (HUMAN F-NAME (CHARLOTTE))
        OBJECT (FOOD TYPE (WEENER)
       AMNT (MULTI))))
```