CS3723 Programming Assignment #3 - LISP Natural Language for Sentences (60 pts) – due date – 2020-04-19

**© Copyright 2020 Larry Clark, this document must not be copied to any other website**

In this assignment, you will use LISP to help understand natural language sentences. Your code will parse the sentence into the parts of speech. If the sentence is incorrect according to our grammar, your **checkSentence** function will return NIL. For extra credit, you will handle more complex sentences.

If you use a solution from another webiste in this program, you will receive a 0 on this assignment, fail this course, and possibly be expelled from UTSA.

**Sample Sentences:**

1. The boy pet the dog
2. The dog ran home
3. Mickey ate a banana
4. The boy threw the ball to the dog
5. The dog sat on the boy
6. Larry taught brilliant students
7. The smelly dog licked the happy boy
8. Mickey ate a large bowl of rancid cheese
9. The strange boy threw a ripe banana at the scared girl
10. Mickey threw the ball to Pluto and Goofy (extra credit 1)
11. Larry and Rocky taught students (extra credit 1)
12. Mickey has a new pet (extra credit 1)
13. The pet barfed (extra credit 1)
14. While Mickey was home comma he made a Halloween costume (extra credit 2)
15. After Goofy left comma many people were sad (extra credit 2)
16. While the strange boy and happy girl took a nap comma the smelly dog licked the new pet (extra credit 2)
17. While Goofy was on vacation comma Pluto barfed on the sleeping pet (extra credit 2)

**Simple sentence grammar:**

* 1. *Sentence := Subject Verb [DirectObj] [Prep IndirectObj]*
  2. *Subject := NounPhrase*
  3. *DirectObj := NounPhrase*
  4. *IndirectObj := NounPhrase*
  5. *NounPhrase := [Article] [Adjective] Noun*

**Notes**:

1. Brackets indicate something that is optional. For example, an *Article* is optional in a *NounPhrase*.
2. The italicized words in black are nonterminal symbols, meaning that they are replaceable with something else.
3. A symbol in black surrounded by quotes is an actual word (e.g., 'and').
4. The symbols in blue within the grammar refer to words. Most of these are defined using the **set\_isa** macro which you must code. (See the **set\_isa** documentation below.) Additionally, you can determine the part of speech for a word by using the **isa** function (which you must code).

* *Verb* values include at least pet, sat, threw, taught, ran, and ate.
* *Prep* values include at least on, of, to, in, and for.
* *Noun* values include at least boy, dog, ball, Mickey, Pluto, Larry, Goofy, and students.
* *Article* values include at least the, a, and an.
* *Adjective* values include at least brilliant, smelly, strange, scared, new, large, rancid, ripe, and sleeping.

1. This program uses three hash tables:

* parse-obj – contains the original sentence, a cursor position, and a value for each part of speech
* word-dict – this provides the part of speech. For extra credit part 1, a word (e.g., pet) can have multiple parts of speech so this could be a list. The key to word-dict is a word

1. Any hard-coding of words other than "AND" or "COMMA" will result in a 0 on this assignment.
2. For extra credit, the grammar will be more complicated. (See below.)
3. Turn in a zip file named LastNameFirstName.zip (no spaces) containing:
   * **p3Lisp.txt** – your LISP source code.
   * Your log of the session (see the setup instructions). This should be a **p3Out.txt**.
   * **Do not have any directories within your zip file.**

**Parsing a Sentence**

There are many parsing approaches; however, you are **required** to use the approach described here.

* You have been provided a parse-obj which is simply a hash table. It contains the sentence to parse, current cursor position (relative to zero), and each part of speech (e.g., subject, verb, directObj, prep, indirectObj).
* You have been provided a word dictionary which is simply a hash table. Each word has a single identified part of speech (unless doing the extra credit). The words will be assigned a part of speech using the **set\_isa** macro. To obtain a word's part of speech, use the **isa** function. The word is the key to the hash-table.
* To access the next word in a sentence, you have been provided a **getToken** function which increments the current cursor position and returns the current token prior to that position change. getToken **must** be used to advance through a sentence. getToken uses **getCursor** (provided) and **setCursor** (provided).
* Example using **getToken** and **isa** (partial code):

(defun checkPrepPhrase (parse objNm)

(prog (prep) ;;; local variable named prep

;;; get a preposition

(setf prep (getToken parse))

;;; it must be a preposition

(if (not (isa preposition 'prep))

(return NIL) )

(if (not (checkNP parse objNm))

(return NIL) )

…

* One difficulty in parsing is when it is necessary to "back up". It is likely that you will need to "back up". In our parsing approach if it is necessary to "back up", save the cursor position before parsing some portion and then use **setCursor** to reset to the old position.   
  Example (assume parse is set to parse-obj)

;;; Assume checkNP uses getToken to get tokens.

;;; If the surrounding code needs to reset the position, it is

;;; done by simply calling setCursor as shown

(setf saveCursor (getCursor parse)) ;;; save current cursor position

(if (not (checkNP parse 'directObj))

(setCursor parse saveCursor)) ;;; back up to the old position

* When saving the parts of speech, which is how you provide results, use (putp *partOfSpeech* parse *value* )

**Sample Results (partial):**

;;;

;;; 1. Subject Verb DirectObj

;;;

(processSentence '(The boy pet the dog ))

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(THE BOY PET THE DOG)

checkSentence returned T

subject= (THE BOY)

verb= (PET)

directObj= (THE DOG)

prep= NIL

indirectObj= NIL

T

;;;

;;; 2. Subject Verb [DirectObj] [Prep IndirectObj]

;;;

(processSentence '(The boy threw the ball to the dog ))

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(THE BOY THREW THE BALL TO THE DOG)

checkSentence returned T

subject= (THE BOY)

verb= (THREW)

directObj= (THE BALL)

prep= (TO)

indirectObj= (THE DOG)

T

(processSentence '(The dog sat on the boy ))

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(THE DOG SAT ON THE BOY)

checkSentence returned T

subject= (THE DOG)

verb= (SAT)

directObj= NIL

prep= (ON)

indirectObj= (THE BOY)

T

(processSentence '(Larry taught brilliant students))

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(LARRY TAUGHT BRILLIANT STUDENTS)

checkSentence returned T

subject= (LARRY)

verb= (TAUGHT)

directObj= (BRILLIANT STUDENTS)

prep= NIL

indirectObj= NIL

T

(processSentence '(Mickey ate a large bowl of rancid cheese))

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(MICKEY ATE A LARGE BOWL OF RANCID CHEESE)

checkSentence returned T

subject= (MICKEY)

verb= (ATE)

directObj= (A LARGE BOWL)

prep= (OF)

indirectObj= (RANCID CHEESE)

T

;;;

;;; 3. Invalid sentences

;;;

(processSentence '(The boy sat on the))

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(THE BOY SAT ON THE)

checkSentence returned NIL

subject= (THE BOY)

verb= (SAT)

directObj= NIL

prep= NIL

indirectObj= NIL

NIL

(processSentence '(Threw the ball at Mickey))

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(THREW THE BALL AT MICKEY)

checkSentence returned NIL

subject= NIL

verb= NIL

directObj= NIL

prep= NIL

indirectObj= NIL

NIL

;;;

;;; 4. Extra Credit #1 compound nouns and words being multiple p-o-s

;;;

(processSentence '(Mickey has a new pet))

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(MICKEY HAS A NEW PET)

checkSentence returned T

subject= (MICKEY)

verb= (HAS)

directObj= (A NEW PET)

prep= NIL

indirectObj= NIL

T

(processSentence '(Mickey threw the ball to Pluto and Goofy))

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(MICKEY THREW THE BALL TO PLUTO AND GOOFY)

checkSentence returned T

subject= (MICKEY)

verb= (THREW)

directObj= (THE BALL)

prep= (TO)

indirectObj= (PLUTO AND GOOFY)

T ;;;

;;; 6. Extra Credit #2

;;;

(processSentence '(While Goofy was on vacation comma Pluto barfed on the sleeping pet ))

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(WHILE GOOFY WAS ON VACATION COMMA PLUTO BARFED ON THE SLEEPING PET)

checkSentence returned T

subject= (PLUTO)

verb= (BARFED)

directObj= NIL

prep= (ON)

indirectObj= (THE SLEEPING PET)

SubConjunction= (WHILE)

SubSubject= (GOOFY)

SubVerb= (WAS)

SubPrep= (ON)

SubIndirectObj= (VACATION)

T

**Parsing a Sentence**

* **You** are **required** to implement the following functions:

(**checkSentence** *parse*) This function checks for a valid sentence according to the grammar above. Assume it is passed a parse-obj (which has already been populated with a sentence) and returns **T** if the sentence is valid; otherwise, it returns **NIL**. Name the parameter **parse** to avoid conflicts. Additionally, checkSentence **saves** the **identified parts of speech** (e.g., (putp 'verb parse (list verb))). checkSentence is invoked by Larry's **processSentence** (see the sample output).

(**resetPartsOfSpeech** *parse partOfSpeech1 partOfSpeech2 ...*)

This function resets the value for each of the specified parts of speech to NIL using **putp**. The first argument is a parse-obj. There are a variable number of parts of speech passed to resetPartsOfSpeech.

Example:  
 (resetPartsOfSpeech parse-obj 'subject 'verb 'prep 'directObj)

(**set\_isa** *partOfSpeech word1 word2 …*)

This macro defines each word in the list of *words* to the specified *partOfSpeech*  in the dictionary (hard code **word-dict**). Use (**putp** *word* **word-dict** *partOfSpeech***)** to put each word in word-dict with the *partOfSpeech.*

Examples:

(set\_isa article a an the)

(set\_isa verb pet sat threw taught ran ate)

In the **extra credit**, a single word could have multiple parts of speech:

(set\_isa verb pet sat threw taught ran ate)

(set\_isa noun pet)

(**isa** *word partOfSpeech*) This function returns T if the specified *word* is that specified *partOfSpeech*; otherwise, NIL is returned.

Example: (isa 'threw 'verb) returns T based on the set\_isa above.

**Larry provided**

* **p3LispDef.txt** – some LISP functions which you need to load prior to loading your p3Lisp.txt
* **p3LispRun.txt** – this will execute your code and should be loaded after you load your code

**Functions Larry Provided (see p3LispDef.txt for more information)**

**(processSentence** *sentence*) - invokes your functions to process a sentence. It invokes resetPartsOfSpeech and checkSentence. You won't use this function directly; instead, p3LispRun.txt uses it.

**(putp** *symbol ht value*) - puts a property about a symbol into a hash table

**(getp** *symbol ht*) - gets a property about a symbol from a hash table

**(getCursor** *parse*)- returns the current cursor position for the specified parse

**(setCursor** *parse cursorPosition*)- sets the current cursor position (usually with a previously saved getCursor position)

**(getToken** *parse*)- gets the next token from the sentence. It also advances the cursor position.

**Extra Credit:**

Inside **p3Lisp.txt**, setf **doingExtra** to one of these values:

* **NIL –** none of the extra credit
* **'EC1 –** only extra credit 1
* **'EC2 –** both extra credits

**To receive extra credit, your submission must not be late and it must meet all requirements.**

**EC#1: (10+100pts/n)** Handle compound nouns and words defined with multiple parts of speech

* Grammar change:
  1. *Sentence := Subject Verb [DirectObj] [Prep IndirectObj]*
  2. *Subject := ComplexNounPhrase*
  3. *DirectObj := ComplexNounPhrase*
  4. *IndirectObj := ComplexNounPhrase*
  5. *ComplexNounPhrase := NounPhrase ['*and' *NounPhrase]*
  6. *NounPhrase := [Article] [Adjective] Noun*
* Your **set\_isa** macro needs to allow some words (e.g., pet) to have multiple parts of speech. This means if **set\_isa** encounters a word that already has a part of speech, the part of speech will become a list.
* Your **isa** function must be able to handle words which have multiple parts of speech

**EC#2: (10+150pts/n)** Handle subordinate clauses

* Must also meet all requirements of EC#1
* Grammar:

1. *Sentence := [SubordinateClause* comma*] Subject Verb [DirectObj] [Prep IndirectObj]*

2. *SubordinateClause := SubConjunction SubSubject SubVerb [SubDirectObj] [SubPrep SubIndirectObj]*

3. *SubSubject := ComplexNounPhrase*

4. *SubDirectObject := ComplexNounPhrase*

5. *SubIndirectObject := ComplexNounPhrase*

6. *SubVerb := Verb*

7. *SubPrep := Prep*

8. *Subject := ComplexNounPhrase*

9. *DirectObj := ComplexNounPhrase*

10. *IndirectObj := ComplexNounPhrase*

11. *ComplexNounPhrase := NounPhrase ['*and' *NounPhrase]*

12. *NounPhrase := [Article] [Adjective] Noun*

Example:

(WHILE MICKEY TOOK A NAP COMMA THE SMELLY DOG LICKED THE NEW PET)

subject= (THE SMELLY DOG)

verb= (LICKED)

directObj= (THE NEW PET)

prep= NIL

indirectObj= NIL

SubConjunction= (WHILE)

SubSubject= (MICKEY)

SubVerb= (TOOK)

SubDirectObj= (A NAP)

**Extra credit notes**:

1. n is the total number of people from all sections who meet **all the requirements** for that particular extra credit.
2. Your submission **must not be late**.
3. Your code must meet all the LISP programming standards.
4. Your code must be properly documented and successfully handle ALL the test cases in p3LispRun.txt

(prog (noun verb))

;;; check for a noun

(setf noun (getCurse parse))

(if (not (isa noun ‘noun))

(return NIL) ) ;;; still need to handle articles and adjectives

;;; record that the noun is the subject (however the actual subject may also have an option article and an optional adjective

(putp ‘subject parse (list noun))

;;; check for a verb

(setf verb (getCursor parse))

(if (not (isa verb ‘verb))

(return NIL) )

;;; record the verb

(putp ‘verb parse (list verb))

(prog (article noun verb))

;;; assume only optional article followed by a noun for subject

;;; check for option article

(setf article (getToken parse))

(if (not (isa article ‘article))

(setf noun article article NIL)

;;; check for a noun

(if (not (isa noun ‘noun))

(return NIL)

;;; record that the noun is the subject (however the actual subject may also have an option article and an optional adjective

(putp ‘subject parse noun (LIST article noun))