COSI135: Fall 2017
Final Project Option: Non-English Parsing

This option is similar to problem set 4, and asks you to implement a version of the Haskell parser `P.hs` for a natural language other than English.

`P.hs` is specifically meant to parse certain English sentences. Your task for this project is to explore what is necessary to create a parser that can parse sentences in another language. It is strongly recommended that you only choose this option if you feel confident enough in a language other than English to explain the structure of that language to non-speakers.

Your starting point should be the starter code provided for PS4 (if you don't have this anymore, it can be provided on request). There are two primary tasks necessary to convert the English-based `P.hs` to a parser for another language: modifying the tree structure generation and building the lexicon.

1. **Modifying tree structure**

   No two natural languages have identical sentence structure. Although many language that have a word order preference share basic word order (English, Spanish, and German all use SVO syntax in many situations), further examination of the languages show structural differences. In Spanish, a direct object pronoun can precede the verb, as in *Yo te amo* ("I love you", lit. "I you love"). In German, compound verbs display the V2 effect, placing the main verb at the end of the sentence, as in *Er hat einen Apfel gegessen* ("He ate an apple", lit. "He has an apple eaten). Other languages display completely different word order preferences (Hindi, Japanese - SOV; Arabic, Gaelic - VSO), so changing the parser for any other language will require changes to the way the parser determines the tree structure and branching of sentences. This will require changes to both `P.hs` and to `Lexicon.hs`. Part of building the lexicon (discussed below) also requires you to implement the possible branching structures of segments that involve that word in the lexicon.

2. **Building the lexicon**

This should be a similar task to what you did for PS4, only you would need to go into further detail for fewer verbs. Rather than adding 100 verbs to the lexicon, you will need to determine the structure for only a few verbs that cover various classes (intransitive, transitive, transitive with indirect object(s), or equivalent verb classes in the language you choose to implement). You will need to consider structural differences between different cases of the same verb usage. For example, if you speak Spanish, you should be able to tell what the difference is between *Juan la ama* ("John loves her") and *Juan ama a María* ("John loves Mary").

You may need to add word categories. For example, the starter code contains a category of `AccOrDat` that will accept a word in the accusative or dative case (in English, this just stands in for the object form of a word, which is a distinction only made in pronouns), but the language you choose may have different forms for the accusative and the dative, or further cases besides, and these would need to be accounted for.

A full implementation of a parser would also account for a robust handling of auxiliaries. Rather than treating periphrastic constructions like "he has gone" as a single unit as was done for PS4 and PS5 (e.g `he has_gone`, we need to acknowledge that in such constructions, the tense and aspect information is distributed across more than one word. The same is true is other languages that use auxiliary con structures. In Italian, *ha mangiato* ("[he/she/it] has eaten"), *mangiato* is a participle form "eaten," while *ha* is an auxiliary (in the 3rd person singular), and together they form the 3rd person singular present perfect form "has eaten." In Hindi, many forms use auxiliaries, including the simple present, as in *vah jātī hãi* – "she goes" – with feminine plural present participle *jātī* and 3rd person plural auxiliary *hãi*.

Some of you may have attempted to handle auxiliaries independently for PS4. If you were successful, you can attempt to adapt that code to the new language. If you didn't try, you can investigate how it might be done. The English auxiliaries "did" and "didn't" are handled in a limited way in `Lexicon.hs`. You can see if you can adapt that code into a more robust handling of auxiliaries.

Your submission should also contain a write-up describing how you attacked this problem, what avenues you explored, and what problems you ran into. There is no length requirement, but you should include some use cases for testing and you should describe your strategy, process, and assumptions in some depth.

NOTES:

- You will probably need to delve a bit into both lexicon-building and tree-structure modification, but how much time you invest in each may be dependent on the language that you choose to implement, so consider that when making your choice. If you would like to focus more on the lexicon side of things, a language like Arabic, with

approximately fifty inflectional forms in the indicative mood active voice alone, may be a poor choice. If you want to focus on tree structure, you may want to avoid free word order languages like Hungarian.

- If you choose a language that is typically written in non-Roman script, please choose a consistent romanization scheme and stick to it. The romanization can include Unicode. GHCI can handle hex and decimal values for unicode characters. `Prelude> putStrLn "\0926"` and `Prelude> putStrLn "\x09be"` both output Greek letter Ξ.

- This, like all the final project options, is very open ended. **It is neither required not expected that you will develop a full parser for even a small subsection of an entire natural language. A parser that can successfully parse a few basic examples will be considered very successful.** No matter how far you do get, you will be graded on the strength of your submission as it stands and on your write up, not in comparison to some ideal. This is a problem that doesn't necessarily have an established answer, and the purpose is simply to explore some of the semantic issues that arise when building a parser for a non-English language using the skills you've learned during the semester.