CENG 242

Homework #2

(Due: April 2nd 2010, Friday 23:55)

In this homework, you will be parsing English sentences. Parsing a sentence is analyzing a sentence into its components. For instance, given the sentence: "I have a great idea.", we can say that "I" is a pronoun, "have" is a verb, "a great idea" is a noun phrase associated with the verb (actually, it is the object), where "a" is an article, "great" is an adjective, and "idea" is a noun. For this mission, we will use some tools borrowed from Formal Languages Theory, which you will encounter in CENG280 in a few weeks.

For parsing a sentence, we can make use a set of rules, known as a grammar. For instance, suppose that we define a sentence in this manner:

```
Sentence -> NounPhrase VerbPhrase
```

meaning that, a sentence is composed of a noun phrase, followed by a verb phrase. Then again, these two substructures can be defined as:

`NounPhrase -> SimpleNoun NounQualifier` (i.e, a noun phrase is composed of something called a "simple noun", followed by a noun qualifier.)

`VerbPhrase -> Verb VerbQualifier` (i.e, a verb phrase is composed of a verb, followed by a verb qualifier.)

Obviously we will need some more rules, let us enumerate all of them at once:

```
SimpleNoun -> Noun | Pronoun | Article SimpleNoun | Adverb SimpleNoun |
Adjective SimpleNoun
```

```
NounQualifier -> E | RelativeClause | PrepositionalPhrase
```

```
VerbQualifier -> E | Adverb VerbQualifier | Adjective VerbQualifier |
Article VerbQualifier | PrepositionalPhrase | Noun | Pronoun
```

```
PrepositionalPhrase -> Preposition NounPhrase
```

```
RelativeClause -> RelativePronoun VerbPhrase
```

(Of course, this is a very simplified set of English rules. Real natural languages, such as English and Turkish are, as you will see, ambiguous, that is, they cannot be parsed in a unique manner.)

The | sign stands for an 'or' operator. This means, a simple noun can be either a noun, or a pronoun, or the combination of an article and another simple noun, etc.

The E sign stands for a null component. It is used for optional structures. For instance, a noun phrase is composed of a simple noun followed by a noun qualifier. This noun qualifier might be either a relative clause,

or a prepositional phrase, or it may not exist at all. In this third case, we say that the noun qualifier matches to E. Examples are:

house that I live in -> noun phrase

simple noun: house (which is a noun)

noun qualifier: that I live in (which is a relative clause)

vs.

house by the sea -> noun phrase

simple noun: house (which is a noun)

noun qualifier: by the sea (which is a prepositional phrase)

vs.

house -> noun phrase

simple noun: house (which is a noun)

noun qualifier: E (does not exist)

Finally, we need to define the list of nouns, adjectives, etc., (i.e, the vocabulary of the language), which is called a lexicon. The lexicon members such as nouns and verbs, which are atomic structures, are called the "terminals" of the language. Composite structures, such as sentences and relative clauses, which are composed of other substructures, are called "nonterminals".

In this homework, you will be given a lexicon, and assuming the grammar is the one given above, you are going to extract a parse tree for a given sentence. The ParseTree type should be defined as follows:

```
data ParseTree  = Empty | Terminal Ident String | NonTerminal Ident
ParseTree | Branch Ident ParseTree ParseTree

data Ident =

                Noun

             |  Pronoun

             |  Article

             |  Adjective

             |  Adverb

             |  Verb

             |  RelPronoun

             |  Preposition
```

```
            |   SimpleNoun
            |   NounQualifier
            |   VerbQualifier
            |   RelClause
            |   PrepPhrase
            |   NounPhrase
            |   VerbPhrase
            |   Sentence
```

(Ident defines the type assigned by the grammar, while Terminal, NonTerminal and Branch constructors help build the tree.)

In English, a parse tree can either be

- empty

- a single Terminal node which has no children. In this case, the "ident" should either be Noun, Pronoun, Article, Adverb, Verb, RelPronoun, or Preposition.

- a node which has a single child. (Such a node is called a NonTerminal.) In this case, the "ident" should either be SimpleNoun, NounQualifier, VerbQualifier, NounPhrase or VerbPhrase.

- a node which has two children. (Such a node is called a Branch.) In this case, the "ident" should either be SimpleNoun, VerbQualifier, RelClause, PrepPhrase, NounPhrase, VerbPhrase, of Sentence.

You will write a function `parse` in the form:

```
parse :: String -> ParseTree
```

Examples:

Assume that the following lexicon is given: (A completely different lexicon will be provided to run your codes. This lexicon is just for the sake of the examples.)

```
liftOfNouns = ["house", "grass", "solution", "problems", "way"]

listOfPronouns = ["i", "you"]

listOfVerbs = ["live", "lives", "know", "knows", "have", "has", "solves",
"am", "is", "are"]

listOfAdjectives = ["great", "happy", "green", "all"]
```

```
listOfAdverbs = ["very"]

listOfPrepositions = ["by", "at", "for"]

listOfRelPronouns = ["that"]

listOfArticles = ["a", "an", "the"]
```

Example 1:

Given the sentence "i am very happy", the extraction of the parse tree can be conducted as follows, using a recursive parser, which starts analyzing the sentence from left:

Step 1-) i am very happy: Sentence

Step 2-) i: NounPhrase

Step 3-) i: SimpleNoun

Step 4-) E: NounQualifier (Meaning that there is no NounQualifier.)

Step 5-) i: Pronoun

Step 6-) am very happy: VerbPhrase

Step 7-) am: Verb

Step 8-) very happy: VerbQualifier

Step 9-) very: Adverb

Step 10-) happy: VerbQualifier

Step 11-) happy: Adjective

Step 12-) E: VerbQualifier (Meaning that there is no more VerbQualifier.)

This should give the following ParseTree:

```
(Branch Sentence (NonTerminal NounPhrase (NonTerminal SimpleNoun (Terminal
Pronoun \"i\"))) (Branch VerbPhrase (Terminal Verb \"am\") (Branch
VerbQualifier (Terminal Adverb \"very\") (NonTerminal VerbQualifier
(Terminal Adjective \"happy\")))))
```

You should notice that the empty branches of the tree (corresponding to E: NounQualifier and E: VerbQualifier) are not viewed. They should be pruned from the tree.

Example 2:

Assume the sentence "the grass by the house is green" is given:

Step 1-) the grass by the house is green: Sentence

Step 2-) the grass by the house: NounPhrase

Step 3-) the grass: SimpleNoun

Step 4-) the: Article

Step 5-) grass: SimpleNoun

Step 6-) grass: Noun

Step 7-) by the house: NounQualifier

Step 8-) by the house: PrepPhrase

Step 9-) by: Preposition

Step 10-) the house: NounPhrase

Step 11-) the house: SimpleNoun

Step 12-) the: Article

Step 13-) house: SimpleNoun

Step 14-) house: Noun

Step 15-) E: NounQualifier

Step 16-) is green: VerbPhrase

Step 17-) is: Verb

Step 18-) green: VerbQualifier

Step 19-) green: Adjective

Step 20-) E: VerbQualifier

Giving the following ParseTree:

```
(Branch Sentence (Branch NounPhrase (Branch SimpleNoun (Terminal Article \"the\") (NonTerminal SimpleNoun (Terminal Noun \"grass\"))) (NonTerminal NounQualifier (Branch PrepPhrase (Terminal Preposition \"by\") (NonTerminal NounPhrase (Branch SimpleNoun (Terminal Article \"the\") (NonTerminal SimpleNoun (Terminal Noun \"house\")))))))) (Branch VerbPhrase (Terminal Verb \"is\") (NonTerminal VerbQualifier (Terminal Adjective \"green\"))))
```

Example 3:

Compare and contrast the sentences: "a solution that solves all is on the way" and "a solution that solves all problems is on the way".

The parsing of the sentence "a solution that solves all is on the way" is as follows:

Step 1-) a solution that solves all is on the way: Sentence

Step 2-) a solution that solves all: NounPhrase

Step 3-) a solution: SimpleNoun

Step 4-) a: Article

Step 5-) solution: SimpleNoun

Step 6-) solution: Noun

Step 7-) that solves all: NounQualifier

Step 8-) that solves all: RelClause

Step 9-) that: RelPronoun

Step 10-) solves all: VerbPhrase

Step 11-) solves: Verb

Step 12-) all: VerbQualifier

Step 13-) all: Adjective

Step 14-) E: VerbQualifier

Step 15-) is on the way: VerbPhrase

Step 16-) is: Verb

Step 17-) on the way: VerbQualifier

Step 18-) on the way: PrepPhrase

Step 19-) on: Preposition

Step 20-) the way: SimpleNoun

Step 21-) the: Article

Step 22-) way: SimpleNoun

Step 23-) way: Noun

resulting in the following ParseTree:

```
(Branch Sentence (Branch NounPhrase (Branch SimpleNoun (Terminal Article
\"a\") (NonTerminal SimpleNoun (Terminal Noun \"solution\"))) (NonTerminal
NounQualifier (Branch RelClause (Terminal RelPronoun \"that\") (Branch
VerbPhrase (Terminal Verb \"solves\") (NonTerminal VerbQualifier (Terminal
Adjective \"all\")))))) (Branch VerbPhrase (Terminal Verb \"is\")
(NonTerminal VerbQualifier (Branch PrepPhrase (Terminal Preposition \"on\")
(NonTerminal NounPhrase (Branch SimpleNoun (Terminal Article \"the\")
(NonTerminal SimpleNoun (Terminal Noun \"way\"))))))))
```

On the other hand, the sentence "a solution that solves all problems is on the way" is as follows:

Step 1-) a solution that solves all is on the way: Sentence

Step 2-) a solution that solves all: NounPhrase

Step 3-) a solution: SimpleNoun

Step 4-) a: Article

Step 5-) solution: SimpleNoun

Step 6-) solution: Noun

Step 7-) that solves all: NounQualifier

Step 8-) that solves all: RelClause

Step 9-) that: RelPronoun

Step 10-) solves all: VerbPhrase

Step 11-) solves: Verb

Step 12-) all: VerbQualifier

Step 13-) all: Adjective

Step 14-) problems: VerbQualifier

Step 15-) problems: Noun

Step 16-) is on the way: VerbPhrase

Step 17-) is: Verb

Step 18-) on the way: VerbQualifier

Step 19-) on the way: PrepPhrase

Step 20-) on: Preposition

Step 21-) the way: SimpleNoun

Step 22-) the: Article

Step 23-) way: SimpleNoun

Step 24-) way: Noun

and gives this ParseTree:

```
(Branch Sentence (Branch NounPhrase (Branch SimpleNoun (Terminal Article
\"a\") (NonTerminal SimpleNoun (Terminal Noun \"solution\"))) (NonTerminal
NounQualifier (Branch RelClause (Terminal RelPronoun \"that\") (Branch
VerbPhrase (Terminal Verb \"solves\") (Branch VerbQualifier (Terminal
Adjective \"all\") (NonTerminal VerbQualifier (Terminal Noun
\"problems\")))))))) (Branch VerbPhrase (Terminal Verb \"is\") (NonTerminal
VerbQualifier (Branch PrepPhrase (Terminal Preposition \"on\") (NonTerminal
```

```
NounPhrase (Branch SimpleNoun (Terminal Article \"the\") (NonTerminal
SimpleNoun (Terminal Noun \"way\")))))))))
```

Constraints:

- "Empty" branches of a tree (corresponding to E: VerbQualifier or E: NounQualifier) should be pruned. See examples.

- You can assume all letters are in lower case.

- DO NOT hard code the lexicon given in the examples into your codes. (THIS IS IMPORTANT FOR GRADING.) You can assume that the lexicon (8 String lists, namely liftOfNouns, listOfPronouns, listOfVerbs, listOfAdjectives, listOfAdverbs, listOfPrepositions, listOfRelPronouns, listOfArticles) will be "known" by the interpreter when your codes will be run. (i.e, in your codes, make use of these lists, but do not define them.)

Show Function:

For checking your ParseTrees, you can use the showParseTree function given below. DO NOT submit the showParseTree function in your codes.

```
showParseTree :: ParseTree -> String

showParseTree (Empty) = "(Empty)"

showParseTree (Terminal Article n) = "(Terminal Article " ++ show n ++ ")"

showParseTree (Terminal Noun n) = "(Terminal Noun " ++ show n ++ ")"

showParseTree (Terminal Verb n) = "(Terminal Verb " ++ show n ++ ")"

showParseTree (Terminal Adjective n) = "(Terminal Adjective " ++ show n ++
")"

showParseTree (Terminal Adverb n) = "(Terminal Adverb " ++ show n ++ ")"

showParseTree (Terminal Preposition n) = "(Terminal Preposition " ++ show n
++ ")"

showParseTree (Terminal RelPronoun n) = "(Terminal RelPronoun " ++ show n
++ ")"

showParseTree (Terminal Pronoun n) = "(Terminal Pronoun " ++ show n ++ ")"

showParseTree (NonTerminal SimpleNoun x) = "(NonTerminal SimpleNoun " ++
showParseTree x ++ ")"

showParseTree (Branch SimpleNoun x y) = "(Branch SimpleNoun " ++
showParseTree x ++ " " ++ showParseTree y ++ ")"

showParseTree (NonTerminal NounQualifier x) = "(NonTerminal NounQualifier "
++ showParseTree x ++ ")"
```

```
showParseTree (NonTerminal VerbQualifier x) = "(NonTerminal VerbQualifier "
++ showParseTree x ++ ")"

showParseTree (Branch VerbQualifier x y) = "(Branch VerbQualifier " ++
showParseTree x ++ " " ++ showParseTree y ++ ")"

showParseTree (Branch RelClause x y) = "(Branch RelClause " ++
showParseTree x ++ " " ++ showParseTree y ++ ")"

showParseTree (Branch PrepPhrase x y) = "(Branch PrepPhrase " ++
showParseTree x ++ " " ++ showParseTree y ++ ")"

showParseTree (NonTerminal NounPhrase x) = "(NonTerminal NounPhrase " ++
showParseTree x ++ ")"

showParseTree (Branch NounPhrase x y) = "(Branch NounPhrase " ++
showParseTree x ++ " " ++ showParseTree y ++ ")"

showParseTree (NonTerminal VerbPhrase x) = "(NonTerminal VerbPhrase " ++
showParseTree x ++ ")"

showParseTree (Branch VerbPhrase x y) = "(Branch VerbPhrase " ++
showParseTree x ++ " " ++ showParseTree y ++ ")"

showParseTree (Branch Sentence x y) = "(Branch Sentence " ++ showParseTree
x ++ " " ++ showParseTree y ++ ")"
```