Ling 472 Lab, November 5, 2004

Revised pseudo code for the (non-probabilistic) CKY algorithm:

```
Create and clear chart[#words, #words]

for i ← 1 to #words
   chart[i, i] ← {α | α → input_i}

for span ← 2 to #words
   for begin ← 1 to #words – span + 1
      end ← begin + span – 1
      for m ← begin to end – 1
         if (α → β1β2 ∈ P ∧
            β1 ∈ chart[begin, m] ∧ β2 ∈ chart[m + 1, end] then
               chart[begin, end] = chart[begin, end] ∪ {α}
```

Step through the (non-probabilistic) CKY algorithm, using this grammar:

```
S  → NP VP
S  → Aux S

VP → V S
VP → V NP
VP → VP PP

NP → Det N
NP → NP PP

NP → Waikiki
NP → Oslo
NP → Kim
NP → snow

PP → P NP
PP → P S

V  → adores
VP → snores

Aux → does
Aux → can
Aux → is
```

```
P  → in
P  → on
P  → before

Det → this
Det → these
Det → the
```

Use this sentence:

```
Snow in Oslo snores
  1   2   3     4
```

First, start out with a chart with the appropriate cells. Each one corresponds to a substring of the input string:

	1	2	3	4
1				
2				
3				
4				

The first loop:

```
for i ← 1 to #words
   chart[i, i] ← {α | α → input_i}
```

This fills in the chart with pre-terminals.

So we go through i = 1 to i = 4; for each of these, we put an element in the corresponding cell in the chart for each preterminal that expands to that input. We end up with a chart that looks like this:

	1	2	3	4
1	**NP**			
2		**P**		
3			**NP**	
4				**VP**

In the next set of nested loops, we build new constituents out of existing ones. Each time we execute the innermost loop, we are looking at two potential daughters and seeing if they form a constituent. If they do, we add that constituent to the appropriate place in the chart. The loops have these variables:

span – The length of the constituent. This varies from 2 to the whole length of the sentence. It starts at two because all the constituents of length one are put into the chart by the initial loop.

begin – The starting position of the constituent. The longer the *span*, the smaller this has to be. This varies from 1 to the length of the input minus *span*, plus 1. So if we're trying to build constituents of length 3, the starting point can be anywhere from 1 to 2.

end – this is fully determined by *begin* and *span*. It is the end of the constituent.

m – The dividing point between the two daughters of the constituent. (Actually, the end point of the first daughter.) This varies from *begin* to *end* minus 1.

In the first iteration of the outermost loop, we're building constituents of length 2, so *span* will be 2. *begin* will range from 1 to 3. *end* will always be 1 more than *begin*. *m* will always be the same as *begin*.

we look at 1,1 (NP) and 2,2 (P) and there is no rule
we look at 2,2 (P) and 3,3 (NP) and add PP to 2,3
we look at 3,3 (NP) and 4,4 (VP) and add S to 3,4

We end up with this table:

	1	2	3	4
1	**NP**			
2		**P**	**PP**	
3			**NP**	**S**
4				**VP**

In the next iteration, we're building constituents of length 3, so *span* will be 3. *begin* can range from 1 to 2. *end* will always be 2 more than *begin*. *m* will range from *begin* to one more than *begin*.

we look at 1,1 (NP) and 2,3 (PP) and add NP to 1,3
we look at 1,2 and don't find anything
we look at 2,2 (P) and 3,4 (S) and add PP to 2,4
we look at 2,3 (PP) and 3,4 (VP) and there is no rule

We end up with this table:

	1	2	3	4
1	**NP**		**NP**	
2		**P**	**PP**	**PP**
3			**NP**	**S**
4				**VP**

In the final iteration, we're building constituents of length 4, so *span* will be 4. *begin* can just be 1. *end* can only be 4. *m* can range from 1 to 3.

we look at 1,1 (NP) and 2,4 (PP) and add NP to 1,4
we look at 1,2 and don't find anything
we look at 1,3 (NP) and 4,4 (VP) and add S to 1,4

We end up with this table:

	1	2	3	4
1	**NP**		**NP**	**NP, S**
2		**P**	**PP**	**PP**
3			**NP**	**S**
4				**VP**