CS-5340/6340, Programming Assignment #2
Due: Friday, September 30 by 11:59pm

Your task is to build a transition-based dependency parser. Your parsing program should be able to run in two different modes, accepting the following command-line arguments:

```
-simulate <sentops_file>
-genops <operators_file> <gold_file>
```

For example, if you are using python, then we should be able to run your program like this:

```
python3 parser -simulate sentops.txt
python3 parser -genops operators.txt gold.txt
```

It is also ok to use Java. The arguments should similarly be accepted on the command-line in the same order.

Given the **-simulate** flag, your program should generate a dependency parse for the sentence in the **sentops_file** by applying the sequence of operators listed in that file.

Given the **-genops** flag, your program should generate the sequence of operators that will produce a correct dependency parse for the sentence in **gold_file**. The **operators_file** will contain the set of possible operators that can be used.

INPUT FILES

The **sentops_file** will be formated as follows:

SENTENCE: <words>
OPERATORS
<op1>
<op2>
etc.

For example, the **sentops_file** might look like this:

SENTENCE: I love NLP
OPERATORS
Shift
Shift
LeftArc_nsubj
Shift
RightArc_dobj
RightArc_root

The **operators_file** will contain a list of transition-based parsing operators, one per line. Each operator will be a single token in one of 2 forms: **LeftArc_<rel>**, or **RightArc_<rel>**. The <**rel**> portion will represent a dependency relation, such as *nsubj*, *dobj*, or *det*. Your parser should apply the LeftArc or RightArc operator and assign the specified dependency relation (<**rel**>) to the resulting arc. **IMPORTANT: the Shift operator should also be used by your parser! The operators file essentially defines the set of dependency relations.**

As an example, the **operators_file** might look like this:

LeftArc_amod
RightArc_amod
LeftArc_aux
RightArc_aux
etc.

The **gold_file** will contain a sentence and the correct ("gold") dependency relations for the sentence. The first line of the file will be "# Sentence". After that, each line of the file will be formatted as:

<**WordID**> <**WordStr**> <**HeadID**> <**Rel**>

The first two columns represent an ID number and string for a word in the sentence. The third column represents the ID of the word's governing head in the dependency parse for the sentence, and the fourth column (Rel) specifies the dependency relation between the word and its governing head.

Assume that ID 0 corresponds to a pseudo-word called ROOT, representing the ROOT node in the dependency graph.

As an example, the **gold_file** might look like this:

# Sentence			
1	I	2	nsubj
2	prefer	0	root
3	the	5	det
4	morning	5	nmod
5	flight	2	dobj
6	through	7	case
7	Denver	5	nmod

This example represents the sentence *"I prefer the morning flight through Denver"*. In the dependency parse for the sentence, the governing head for *I* is word 2 (*prefer*) and the dependency relation between *I* and *prefer* is "nsubj" (i.e., $prefer \xrightarrow{nsubj} I$).

Word 2 (*prefer*) has Word 0 as its governing head (corresponding to the pseudo-word ROOT) and the dependency relation between *prefer* and *ROOT* is "root" (i.e., $ROOT \xrightarrow{root} prefer$).

SIMULATING A DEPENDENCY PARSE

Given the -simulate flag and a **sentops_file**, your program should apply the **Transition Parsing Algorithm** to the given sentence using the given operator sequence. The operator sequence in the **sentops_file** represents what a perfect Oracle would dictate. In other words, applying that operator sequence to the sentence should produce a correct dependency parse.

OUTPUT SPECIFICATIONS

The output from your program for the dependency parse simulation should consist of three parts: (1) Print the sentence and operators found in the **sentops_file**, (2) Print the operations in the order in which they occur, and (3) Print the final set of dependency relations, in the order that they were produced. Please format your output exactly as shown below, including exactly one space between the words in the sentence and one space between the operators (they should all be printed on a single line).

SENTENCE: $word_1$ $word_2$ etc.
OPERATORS
$operator_1$ $operator_2$ etc.

PARSING NOW
$operation_1$
$operation_2$
etc.

FINAL DEPENDENCY PARSE
$relation_1$
$relation_2$
etc.

Each *operation* should be printed in one of two forms:

If the operator is Shift, then print:
"Shifting <WordStr>(<WordID>)"

If the operator is LeftArc or RightArc, then print:
"<Operator> to produce: *relation*"

Each *relation* should be printed as follows, where HeadStr is the string of the governing head and HeadID is its ID:

<HeadStr>(<HeadID>) - - <Rel> - - > <WordStr>(<WordID>)

As an example, here is the correct trace output for the **sentops_file** shown on Page 1:

```
SENTENCE: I love NLP
OPERATORS
Shift Shift LeftArc_nsubj Shift RightArc_dobj RightArc_root

PARSING NOW
Shifting I(1)
Shifting love(2)
LeftArc_nsubj to produce: love(2) -- nsubj --> I(1)
Shifting NLP(3)
RightArc_dobj to produce: love(2) -- dobj --> NLP(3)
RightArc_root to produce: ROOT(0) -- root --> love(2)

FINAL DEPENDENCY PARSE
love(2) -- nsubj --> I(1)
love(2) -- dobj --> NLP(3)
ROOT(0) -- root --> love(2)
```

Your program should print to standard output, not write to a file.

GENERATING OPERATORS FOR DEPENDENCY PARSING

Given the **-genops** flag, your program should generate a sequence of operators that will produce a correct dependency parse for the sentence in **gold_file**. This should follow the process described in the lecture slides for creating training instances to train a machine learning Oracle. The **operators_file** will contain the set of possible operators that can be used, plus your program should use the Shift operator.

OUTPUT SPECIFICATIONS

The output from your program when generating an operator sequence should consist of three parts: (1) Print the sentence and gold dependency relations found in the **gold_file**, (2) Print information about the operators that are produced, as they are generated, and (3) Print the final operator sequence. Please format your output exactly as shown below, including exactly one space between the words in the sentence:

SENTENCE: $word_1$ $word_2$ etc.
GOLD DEPENDENCIES
$relation_1$
$relation_2$
etc.

GENERATING PARSING OPERATORS
$production_1$
$production_2$
etc.

FINAL OPERATOR SEQUENCE
$operator_1$
$operator_2$
etc.

Each *relation* should be printed as follows, where HeadStr is the string of the governing head word and HeadID is its ID:

<HeadStr>(<HeadID>) - - <Rel> - - > <WordStr>(<WordID>)

Each *production* should be printed in one of two forms:

If the operator is Shift, then simply print Shift

If the operator is LeftArc or RightArc, then print:
<Operator> to produce: *relation*

As an example, here is the correct output for *"I love NLP"*

```
SENTENCE: I love NLP
GOLD DEPENDENCIES
love(2) -- nsubj --> I(1)
ROOT(0) -- root --> love(2)
love(2) -- dobj --> NLP(3)

GENERATING PARSING OPERATORS
Shift
Shift
LeftArc_nsubj to produce: love(2) -- nsubj --> I(1)
Shift
RightArc_dobj to produce: love(2) -- dobj --> NLP(3)
RightArc_root to produce: ROOT(0) -- root --> love(2)

FINAL OPERATOR SEQUENCE
Shift
Shift
LeftArc_nsubj
Shift
RightArc_dobj
RightArc_root
```

Your program should print to standard output, not write to a file.

GRADING CRITERIA

We will run your program on new input files to evaluate the generality and correctness of your code. **So please test your program thoroughly!** Even if your program works perfectly on the examples that we give you, this does not guarantee that it will work perfectly on different test cases.

Please make sure that your program conforms exactly to the input and output specifications, or a penalty will be deducted! Programs that do not conform to the specifications are a lot more difficult for us to grade.

SUBMISSION INSTRUCTIONS

On CANVAS, please submit an archived (.tar) and/or gzipped (.gz) file containing:

1. The source code for your `parser` program. Be sure to include all files that are needed to compile and run your program!
2. A README file that includes the following information:
   - how to compile and run your code
   - which CADE machine you tested your program on (this info may be useful to us if we have trouble running your program)
   - any known bugs, problems, or limitations of your program
3. On the CADE machines, run your `parser` program using the data files on Canvas and submit its output. Specifically, you should submit two files:

   sentops1-output.txt : save the output from running your program as:
   ```
   parser -simulate sentops1.txt
   ```

   gold1-output.txt : save the output from running your program as:
   ```
   parser -genops operators.txt gold1.txt
   ```