Homework 8 — ML — due Monday 29 April

Total number of points available on this homework is 260. Full credit is equivalent to 100 points.

8.1 Grammar specification language (100pts)

The grammar of the language used to specify grammars is specified here in terms of itself:

```
Grammar -->  Symbols_List;
Symbols_List --> Symbol Symbols_Separator Symbols_List
               | Symbol;
Symbol --> Identifier  Right_Arrow  Productions_List;
Productions_List --> Production
                     Productions_Separator Productions_List
                   | Production;
Production --> Identifiers_List;
Identifiers_List --> Identifier  Identifiers_List |
```

About the specification language: The terminal symbols in the grammar specification language are identifiers, the right arrow, and the production separator. Identifiers consist of digits, letters, and the underscore. The right arrow is written `-->` or `::=`, the productions separator is written `|`, and the symbols separator is written `;`. White space is allowed anywhere between terminal symbols, but is only required between two adjacent identifiers.

About the language specified: Identifiers that appear on the left-hand side of right arrows are non-terminals. Any other identifiers are assumed to be terminals. Note that we do *not* use a visible representation for the empty production (such as ε).

1. (10pts) Devise ML types to represent (context-free) grammars; name the main type `grammar`. Hint: A good grammar representation captures the symbols (terminals and nonterminals), the distinguished start symbol, and the productions. Recall that a production has a nonterminal symbol on the left, and a string of terminals and nonterminals on the right. Here is a possible representation:

   ```
   type identifier = string
   type expansion = identifier list
   type production = identifier * expansion
   type grammar = {
                   startSymbol: identifier,
                   symbolNames: identifier list,
                   nonterminalNames: identifier list,
                   terminalNames: identifier list,
                   productions: production list
                  }
   ```

2. (10pts) Write a function `pretty_print_grammar: grammar -> unit` to print out a grammar; this will help you to debug the next stage.

3. (60pts) Write a function `parse_grammar: string -> grammar` to parse a grammar specification into internal form. Split the task into a scanner and a parser proper. Syntax errors should raise exceptions (both in the scanner and in the parser), which should be handled in the outermost scan and parse functions, and appropriate error messages should be printed. Hint: There are many ways to write the parser. The easiest is probably to follow the structure of the parser presented in class.

4. (20pts) Package your code using the ML module language. Write a structure `Grammar` with an appropriate signature so that it provides the type `grammar`, the functions `parse_grammar` and `pretty_print_grammar`, and nothing else. (You may write auxiliary structures for scanning and parsing if you like.)

8.2 Extra credit: A slightly different language (80pts)

This exercise is a continuation of Exercise 8.1.

Consider a slightly modified grammar for specifying grammars:

```
Grammar -->  Symbols_List
Symbols_List --> Symbol  Symbols_List
               | Symbol
Symbol --> Identifier  Right_Arrow  Productions_List
Productions_List --> Production
                     Productions_Separator Productions_List
                   | Production
Production --> Identifiers_List
Identifiers_List --> Identifier  Identifiers_List |
```

The difference from Exercise 8.1 is that there is no longer a visible separator between symbol definitions. Write a function `parse_grammar': string -> grammar` to parse a grammar specification into internal form. You can reuse the scanner from Exercise 8.1, but you may discover that you can't reuse the parser (depending on how you wrote it).

8.3 Extra credit: Grammar analysis (80pts)

This exercise is a continuation of Exercise 8.1; it does not depend on Exercise 8.2.

An important question about a grammar is whether it allows predictive top-down parsing. To answer that question, one must first determine which terminal symbols may appear at the head of sentential forms derived from each nonterminal symbol.

1. (70pts) Write a function `first_set: grammar -> string -> string list` with the following purpose: `first_set G S` is the list of terminal symbols of the grammar `G` that can appear as the first symbol in some string derived from the string `S`, where `S` is a symbol of the grammar `G`.

2. (10pts) Package the function `first_set` into a structure `GrammarAnalysis`.

How to turn in

Turn in your code by running *~dmykola/handin your-file* on a regular UNM CS machine. You should use whatever filename is appropriate in place of your-file. You can put multiple files on the command line, or even directories. Directories will have their entire contents handed in, so please be sure to clean out any cruft.

Remember to submit extensive tests of your programs!

Homework must be accompanied by the following statement: *"I pledge my honor that in the preparation of this assignment I have complied with the University of New Mexico Board of Regents' Policy Manual, including Section 4.8, Academic Dishonesty."* The manual is available at `http://www.unm.edu/~brpm/index.html`.