In this section exercise, you will use syntax-directed translation to make a syntax-highlighter for the lambda calculus, a simple programming language with application (which we write as '@'), abstraction ('lambda'), and variables.

Problem 1

We've started you off with a grammar in lambdaPretty.grm:

```
%left '@'
%%

E -> VAR                          %{ return n1.val; %}
  | 'lambda' VAR '.' E            %dprec 2 %{ return '&lambda;' + n2.val + ' . ' + n4.val; %}
  |  E '@' E                      %dprec 1 %{ return n1.val + ' ' + n3.val; %}
  ;

VAR -> /[A-Za-z]/ ;
```

This grammar pretty-prints the program with a Greek lambda. It drops '@' for application (which is there to make the grammar easier to disambiguate) and shows application instead as two expressions next to each other.

Add parentheses to the grammar. Your result should also have parentheses. When you are done, you should be able to run the following commands:

```
node mainRE2.js lambda.lam lambdaPretty.grm > lambdaPretty.html
node mainRE2.js lambda2.lam lambdaPretty.grm > lambdaPretty2.html
```

Your output files should be the same as lambdaPrettyExample.html and lambdaPrettyExample2.html, respectively.

Problem 2

Next, you will add some syntax highlighting for function applications.

If you look at the source for lambdaHighlightExample.html, you will see the following:

```
<span class="id8" style="">x</span>
<span class="id9" style="">x</span>
<script>
   $(".id8").hover(function() {
      $(".id8").css("background", "green");
      $(".id9").css("background", "pink");
   }, function(){
      $(".id8").css("background", "");
      $(".id9").css("background", "");
    });
</script>
</span>
```

Your goal is to produce similar code automatically with your grammar (the IDs can be different). At the end of the end of this step, when you hover over the first element of a function application, the first element will show up in green, and the second element will show up in pink.

Step 1: Generalize this code into an SDT rule. In an empty file or on scratch paper, write out what this code would look like for some values **n1.val** and **n3.val**. You can generate unique IDs by calling `var uid = genUID();`

Step 2: Add the generalized code to your grammar.

You can add code directly into your rule.

For example, the following code would wrap a variable in a span:

```
E -> VAR                    %{ var uid = genUID(); return '<span class="' + uid + '">' + n1.val + '</span>'; %}
```

Similarly, you can add JavaScript to the end of any rule to also generate JavaScript.

When you are done, generate your HTML files again. You should see code similar to the example. Hover over the elements to see the syntax highlighting in action.

Problem 3

In this problem, you will add syntax highlighting for matching parentheses. When you hover one parenthesis, it and the matching parenthesis should be highlighted in orange. Look at `lambdaHighlightExample.html` and `lambdaHighlightExample2.html` for examples.

Step 1: Write the code you want to generate. In an empty file or on scratch paper, write out what this code would look like. Remember that you want to highlight both elements in the pair, so it may help to use the same class IDs for a given pair.

Step 2: Add this code to your grammar.

When you are done, generate your HTML files again. Hover over parenthesis to see the syntax highlighting in action.

Problem 4

Finally, you will add syntax highlighting for lambda expressions. When you hover over a lambda, both the variable and the expression bound by the lambda should be highlighted in cyan.

When you are done, generate your HTML files again. Your files should now work the same way the examples work.

Questions to Consider

If you have extra time, consider some the following questions:
1. What happens if you switch `%dprec` for lambda and function application?
2. What happens if you get rid of `%left '@'`? What happens if you change it to `%right '@'`?
3. How could you extend this to highlight all uses of a bound variable within a lambda?
4. How else could you make this syntax highlighting more useful? Assume you can propagate information through the tree.
5. How could you use this grammar to generate an AST?