All students should read Chapter 5 through to §5.3 of the textbook in preparation for the next lecture.

Distribute the following questions across the members of your group. You will share your solutions (and most importantly the *method* of your solutions) during the next lecture period. Divide up the questions so that **each** question has at least two solutions from different group members.

For all of these questions begin (of course) with your collective code base from `lga-cfg-code.pdf` ; by now you should have chosen two implementations for your group to further develop. Each group member should implement at least one of the three critical algorithms in one of the code bases; and it **shouldn't** be an implementation you created.

If you are the original author of one of the chosen implementations, **you must** do question 1!

It sounds like these development tasks are difficult to parallelize (for instance, all the algorithms depend on *derivesToLambda*()). If you are the first one working on a solution to $followSet()$ just choose **very simple** grammars to develop with and "stub" out calls to $firstSet()$ and *derivesToLambda*():

A "dead simple" grammar:

$$
\begin{array}{lcl}
S & \rightarrow & A\ B\ C\ \$ \\
 & | & B\ A\ C\ \$ \\
A & \rightarrow & a \\
B & \rightarrow & \lambda \\
 & | & b \\
C & \rightarrow & c \\
 & | & q\ r\ s
\end{array}
$$

```
# dummied out for a dead-simple grammar
procedure derivesToLambda( L∈N, stack T )
  if ( L is S ) return false
  if ( L is A ) return false
  if ( L is B ) return true
  if ( L is C ) return false

# dummied out for a dead-simple grammar
procedure firstSet( Xβ, set T )
  if ( Xβ[0] is S ) return {a,b}
  if ( Xβ[0] is A ) return {a}
  if ( Xβ[0] is B ) return {b}
  if ( Xβ[0] is C ) return {c,q}
  # X is a terminal
  return {X}
```

1. **For original authors** — Quickly! squeeze some documentation into your code and publish it for your group (git? mercurial? rcs[1]?). It doesn't have to be prolific, but it should help them see how you've organized the CFG data and how any pre-existing helper routines are called.
2. Implement logic for the *derivesToLambda*() as described in chapter 4 (or here: `derivesToLambda.pdf`)
3. Implement logic for the $firstSet()$ as described in chapter 4 (`firstSet.pdf`).
4. Implement logic for the $followSet()$ as described in chapter 4 (`followSet.pdf`).

Don't be fooled, last question on the next page!

[1] ha!

5. Prepare at least two sample grammar definition files (you can borrow from your group's lga-cfg-code.pdf solution of course) that include examples of λ rules, recursive rules, and non-terminals with multiple production rules, and at least one multiply-recursive rule:

$$A \rightarrow \mathrm{x}\, A\, B\, \mathrm{y}\, A\, \mathrm{z}$$

Document **by hand** their correct results for the algorithms in 2–4.

These don't have to be CFGs for **real computer languages**, keep the terminals and non-terminals easy and the actual CFG text file easy to read (don't string together long | production alterations). Here is an example of what I mean:

```
S -> A B C $
A -> w A C A t
   | F
   | lambda
B -> m n o p
C -> B A z
   | A B C
E -> s | t
F -> z E
   | lambda
```

While I won't micro-manage your distribution of effort, the following might be a good starting point for "negotiations."

Groups of four students should really focus on just one code base for their CFG code.

Groups of five,
student A wrote code base A_c,
student B wrote code base B_c

Student A: Do questions 1, 4 for B_c
Student B: Do questions 1, 4 for A_c
Student C: Do questions 2 and 3 for A_c
Student D: Do questions 2 and 3 for B_c
Student E: Do question 5 and assist or test for others.

Groups of four,
student A wrote the code base

Student A: Do questions 1, 5 and assist or test for others.
Student B: Do questions 2 and 3
Student C: Do questions 3 and 4
Student D: Do questions 4 and 2