# Chapter 3

1. Syntax vs Semantics
2. Lexeme vs Token
3. BNF & extended BNF notations fundamentals
4. Derivations (left and right)
5. Parse Trees
6. Attribute Grammars (cursory, at least be able to describe the following):
   1. Attributes (types and meanings : synthesized, inherited)
   2. Predicate functions
   3. Semantic functions

# Chapter 4

1. Lexer vs Parser
2. Parser categories, meanings, and implementation
   1. Top-down parsers (LR)
   2. Bottom-up parsers (LL)
3. Left-recursion in LL grammar
   1. How to remove
   2. Why is it a problem
4. Pairwise disjointness
   1. What is it?
   2. How to remove it
5. Basic steps and how-to of LR parser (bottom-up)
   1. Creating closures for all grammar rules (DFA)
   2. Transition and parse tables
   3. Parsing input string using PDA

# Chapter 5

1. Variables and attributes
2. Binding
   1. When it happens
   2. What it is
   3. Static vs dynamic
   4. Type binding
      1. Explicit/implicit vs dynamic
   5. Storage bindings & lifetimes
      1. Static
      2. Stack-dynamic
      3. Explicit/implicit heap-dynamic
3. Scope
   1. Scope vs lifetime
   2. Static vs dynamic scope
   3. Referencing environment

# Chapter 6

1. Data type definition
2. Primitive data type vs other
3. Non-scalar data types (cursory)
   1. Be able to recognize the different non-scalar types and at least compare them to arrays on a very basic level
4. Arrays
   1. Jagged vs rectangular
   2. How to calculate linear address given multi-dimensional
      1. Ex: what would linear offset be for array1[5][7] in a 0-based (index starts at 0) array for row-major/column-major array. If you counted all of the cells preceding the one that you wanted and including the actual cell you want and possibly shift by the starting index (0 or 1), this would be the address. There is a formula for this that is described in the book.
5. Pointers (cursory)
   1. Major operations and how-to
   2. Heap management (garbage collection schemes)