9 13

1. Scanners
   1. Groups char into tokens removes comments, etc…
   2. Ad hoc scanners- fast efficient
   3. Automatically generated- generates from tables, etc…
2. Fa scanner
   1. Sometimes have to peek ahead 1 or 2 characters
3. Table driven scanning
   1. Entries say what to do in certain situations
   2. Second table indicates for each state whether we might be at th ened of a token
   3. Issues- errors handling invalid tokens
   4. Ex
      1. Letter = (a|b…)
      2. Digit = (1|2…)
      3. Id = letter(letter|digit)\*
      4. Tables
         1. Char class
            1. Value letter letter digit other

A-z A-Z 0-9 other

* + - 1. Next state
         1. Class s0 s1 s2 s3
         2. Letter s1 s2…
         3. Digit
         4. Other

1. Context free grammers
   1. Every production is N->O
      1. N is a single non terminal
      2. O is a string of non terminals and terminals
   2. Allows stack
   3. Need stack to keep track of recursion
   4. Advantages
      1. Pricise snytatic specification
      2. Easy to understand
      3. Easy to maintain and add to
      4. Can automatically construct efficient parsers
      5. Gives structure to language
      6. Syntax directed translation
      7. No look ahead O(n^3) problem
2. Derivations
   1. Deriving stuff-> sentential forms
   2. Types
      1. Leftmost/rightmost/ambiguous
         1. Differs in which non terminal side you replace first
3. 2 types of parsers
   1. LL- read left to right, leftmost derivation
      1. Simpler, clearer
   2. LR- left to right, rightmost derivation
      1. Larger, intuitive
      2. Can be automatically generated more easily
4. Parse trees
   1. Ambiguous- 2 dif parse trees
   2. Canonical form grammar- simplify applications and analysis
5. Unambiguous grammar
   1. Captures precedence