COP3402 Exam#1 Topics and Promises Spring 2011

1. Phases of a compiler --
   Analysis: Lexical, syntactic, semantic,
   Synthesis: int. code gen., code gen., code opt.
   Helpers -- Table management, Error Handling
   Machine independence of all but code gen. and certain parts of optimization.
   Cost of general parsing (O(2^N) with backtrack, O(N^3) with CKY algorithm, want O(N).)

2. Lexical Analysis:
   Wirth's lexical analyzer for Pascal-S. Your lexical analyzer for Pascal-S.
   Common errors in lexical analysis -- Comments, inconsistency on look-ahead.
   Transition diagrams, regular expressions.
   Elementary concepts of deterministic and non-deterministic FSAs.
   FLEX as a basis for lexical analysis.

3. Grammars, languages, recognizers -- Chomsky hierarchy:
   Phrase Structured grammars (PSG); Turing Mach.
   Context Sensitive grammars (CSG); CSL; LBAs
   Context free grammars (CFG) ; CFL ; PDAs
   Regular grammars (Right Linear); Regular langs. ; FSAs
   Non-determinism in recognizers.

4. Basic idea behind context free grammars and syntax-directed translations.
   Leftmost, rightmost derivations and parse trees.
   Top-down versus bottom-up approaches to parsing.
   Non-recursive predictive parsing: LL(1) parsing. LL(k)>LL(k-1).
   Ambiguity, distinction between ambiguous grammar and ambiguous language.
   Simple ambiguous expression grammar.
   Relation of rightmost to Bottom-up and leftmost to Top-down.
   Decision problems about grammars, e.g., the meaning of the unsolvability of the ambiguity problem.
   More complex, non-ambiguous expression grammar.
   Concepts of abstract syntax tree
   Translation of infix to prefix by syntax directed translation
   Annotated parse trees (attached attributes).
   Translation schemes (essentially procedural with embedded actions.)
   Carrying out simple syntax directed definitions by translation schemes.

5. Top-down Parsing -- looking at mini compilers.
   Control structure parsing in Pascal-S.
   Backtrack and its problems.
   Predictive parsers require no backtrack.
   Elimination of left recursion.
   Left Factoring
   Extended Bachus-Naur Form (EBNF)
   Railroad charts (syntax graphs).
   Recursive descent is an example of predictive parsing.
   One procedure per non-terminal.
   Compute FIRST and FOLLOW to decide which rule to use.
   Abstract stack machines.
   Adding syntax-direction translation to a recursive descent parser.

6. Top Down Parsing: Stack and Parse table
   Basic idea is to start with S on stack and end up with empty stack when input exhausted
   This approach runs rules forward to produce a match for input
   General technique --
      Push S onto stack
      Basis is top of stack and next input.
      Repeat
         If tos=input then pop and read
         If tos is a non-term then consult table entry for this non-term/input pair
      Until input exhausted and stack is empty
   Want to avoid conflict by just looking at next token
   Computation of FIRST and FOLLOW.
   Creation of parse table.
   LL(1) grammars. No multiple entries in parse table OR
      Said differently A -> x | y implies FIRST(x) intersect FIRST(y) is null and
      if y is NULLABLE, then FIRST(x) intersect FOLLOW(y) is null.
   Parse Table and parsing algorithm.

7. Bottom Up Parsing: Shift/Reduce with Stacks.
   Basic idea is to start with empty stack and end up with S as only element in stack when input exhausted
   This approach runs rules backwards, shifting input into stack to help form right hand sides of rules
   General technique --
      Empty stack
      Basis is top of stack and next input.
      Repeat
         If tos == right side of some rule, we may replace rhs with symbol on left
         We may always shift next input symbol to tos
      Until input exhausted and (stack contains just S OR cannot find a rule matching tos
   Want to avoid conflict (reduce/reduce or shift/reduce or both)
   LR(1) grammar allows us to disambiguate with one symbol look-ahead

8. Bottom Up Parsing of arbitrary CFL (CKY)
   Start with CF grammar; convert to Chomsky Normal Form (CNF)
   For input of length n, $a_1a_2\ldots a_n$, build n by n upper triangular matrix
   Populate first row so that A is in the j-th column if $A \rightarrow a_j$.
   Meaning of i-th row, j-th column, when j>1 is:
      Place all A in this slot such that $A \Rightarrow^* a_j \ldots a_{j+i-1}$
   If the slot in column 1, row n contains S then $S\ A \Rightarrow^* a_1a_2\ldots a_n$ is verified
   The key is to use Dynamic Programming in which the j-th row is not processed until all prior rows are done. Row 1 is easy. The notes show how to fill out others.

   Promises:
   1. An expression grammar that incorporates precedence and associativity.
   2. Distinction between languages and grammars in a particular class.
   3. Ambiguity
   4. FLEX type answer to a regular expression problem.
   5. EBNF / Railroad chart question
   6. Creation of a recursive descent parser for some simple construct.
   7. Creation of FIRST, FOLLOW and an LL(1) parse table.
   8. Removal of left recursion and common prefixes.
   9. Bottom-Up and Top-Down stack manipulation