Concepts of Programming Languages, CSCI 305, Fall 2021
Review for final, Dec. 8

New items are in yellow.

Introduction, Chapter 1

- Know various ways that languages are categorized
- Know the difference between procedural/imperative languages and declarative/non-imperative languages and some examples of each.
- Be able to contrast the process of assembling, compiling, and interpreting, and that modern languages do a combination
- Know how the following fit into language translation
    - scanner, parser, semantic analysis and intermediate code generation
    - character stream, token stream, parse tree, abstract syntax tree
- Know the difference between the front and back end of language translation and how one front end can be used with different back ends, and vice-versa.

Programming Language Syntax, Chapter 2, pages 43-69

Section 2.1 Specifying Syntax: Regular Expressions and Context-Free Grammars
Section 2.2 Scanning

- Be able to explain where errors will be captured in the compilation process: lexical analysis (scanning), syntactic analysis (parsing), semantic analysis, dynamic semantic analysis or compiler can't catch.
- Know the difference between syntax and semantics, and be able to give examples of each in the context of program translation.
- Know the tasks of a lexical analyzer.
- Know the difference between a lexeme and a token.
- Know the definition of regular expressions
- Be able to convert:

Language description ⇒ Reg-ex
Reg-ex ⇒ Language description

- Given pattern descriptions for tokens, be able to give regular expressions and productions for recognizing them.
- Know the definition of NFAs and DFA
- Be able to do the following conversions using the method given in the text and discussed in class:

Reg-ex ⇒ NFA
NFA ⇒ DFA

- Given pattern descriptions for tokens, be able to create an NFA, convert it to a minimal DFA and give a table to recognize the tokens using table-driven lexical analysis.
- Know the definition of grammar and be able to translate from a description of a language to a context-free grammar for it.
- Be able to describe the format of context-free languages using BNF.

- Know that BNF for grammars typically includes recursion, whereas BNF for tokens doesn't include recursion.
- Know what it means for a context-free grammar to be ambiguous and why ambiguous grammars are avoided.
- Be able to enforce operator precedence and associatively in context-free grammars.
- Be able to give a leftmost and right-most (canonical) derivation of a string, given a grammar.
- Given the parsing code for a recursive descent parser or for a table-driven parser, a grammar, and a piece of code to parse, be able to show the process of parsing the program.
- Be able to describe how an LL parser parses a program.
- Given an LL(1) grammar, be able to give the predict sets for each production, create a parsing table, and walk through the process of parsing a simple program in the language of the grammar.
- Know the meaning of the term LL parser, know its properties, be able to recognize when a simple grammar is LL(1), and be able to convert simple grammars to LL(1) grammars (this will not always be possible).

Names, Scopes and Bindings, Chapter 3 (Address space on page 793)

- Given a binding between two objects (e.g. name to storage location, or value to a variable) know when they typically occur, or be able to describe the situations in which they would occur at different times.
- Be able to give examples of bindings which occur at language design time, language implement time, compile time, link time, load time, or running time.
- Know what is meant by the scope of a binding.
- Know the difference between static and dynamic bindings and advantages/disadvantages of each.
- Know what is meant by object lifetime, and the key events in the life of an object
- Know an example of when a binding extends beyond the lifetime of an object.
- Know the three primary mechanisms for managing the lifetime of an objects in memory and be able to describe each.
- Know a typical organization of memory and where the above categories of variables are stored within that organization.
- Be able to hand execute code, telling the output when static binding is used and the output when dynamic binding is used.
- Know the definition of orthogonality, know language examples that are and are not orthogonal, and be able to argue why orthogonality is beneficial in programming languages.
- Know the difference between first, second and third class objects and language examples of each.
- Know the difference between deep and shallow bindings.
- Know what is meant by a referencing environment or a closure.

Semantic Analysis, Chapter 4, pages 175-196

- Know the difference between static and dynamic semantics in programming languages.
- Know ways attribute grammars can be used, be able to create simple attribute grammars and be able to determine the purpose of a simple attribute grammar.
- Know the difference between intrinsic, synthesized and inherited attribute grammars.

Data Types, Chapter 7, page 297-304

- Know the purpose of type systems
- Know what is typically meant by strong and weak type systems, but that definitions differ
- Know examples of language characteristics that are typically consider strong, and other examples that are typically considered weak
- Know what is meant by static and dynamic type systems
- Know examples of language characteristics that are static and dynamic
- Know what is meant by “type inference” and know what is meant by “duck typing”

Functional Languages, Chapter 11, 535-550, 581-584

Section 11.1 Historical Origins
Section 11.2 Functional Programming Concepts
Section 11.3 A Bit of Scheme
Section 11.8 Functional Programming in Perspective
Section 11.9 Summary and Concluding Remarks

- Know that Scheme came from LISP which stands for LISt Processing (defined by John McCarthy in 1958)
- Know the meaning of referential transparency and that pure functional languages have referential transparency
- Know that Scheme operates on lists and know how these lists are stored
- Know the following Scheme built-in functions:
  car, cdr, cons, list, append, length, define, lambda, if, cond, member, assoc, let, begin
- Be able to write simple programs in Scheme

Logic Languages, Chapter 12, pages 591-604, 612-617

- Know the theoretical basis of logic languages, how they compute and their strengths
- Know the format of a Horn clause, along with the meaning of a “bodiless” Horn clause.
- Know that Prolog programs consist of a database of facts and rules, and queries made against that database
- Be able to trace a Prolog program given the database and a query
- Given Prolog code, be able to describe what it does
- Be able to write simple Prolog programs