Programming Language Syntax

1. What is the difference between syntax and semantics?

2. What are the three basic operations that can be used to build complex regular expressions from simpler regular expressions?

3. What additional operations (beyond the three of regular expressions) is provided in context-free grammars?

4. What is Backus-Naur form? When and why is it devised?

5. Name a language in which indentation affects program syntax.

6. When discussing context-free languages, wht is a derivation? What is a sentential form?

7. What is the difference between a right-most derivation and a left most derivation?

8. What does it mean for a context-free grammar to be ambiguous?

9. What are associativity and precedence? Why are they significant in parse trees?

10. List the tasks performed by the typical scanner.

11. What are the advantages of an automatically generated scanner, in comparison to a handwritten one? Why do many commercial compilers use a handwritten anyway?

12. Explain the differences between deterministic and nondeterministic finite automata. Why do we prefer deterministic variety for scanning?

13. Outline the constructions used to turn a set of regular expressions into a minimal DFA.

14. What is the "longest possible toke" rule?

15. Why must a scanner sometimes "peek" at upcomming characters?

16. What is the difference between a keyword and an identifier?

17. Why must a scanner save the text of tokens?

18. How does a scanner identify lexical errors? How does it respond?

19. What is a pragma?

20. What is the inherent "big-O" complexity of parsing? What is the complexity of parsers used in real compilers?

21. Summarize the difference beteen LL and LR parsing. Which one of them is also called "bottom-up"? "Top-down"? Which one is also called "predictive"? "Shift-reduce"? What do "`LL`" and "`LR`" stand for?

22. What kinds of parser (top-down or bottom-up) is most common in production compilers?

23. Why are right-most derivations sometimes called canonical?

24. What is the significance of the "1" in `LR(1)`?

25. Why might we want (or need) different grammars for different parsing algorithms?

26. What is an epsilon production.

27. What are recursive descent parsers? Why are they used mostly for small languages?

28. How might a parser construct an explicit parse tree or syntax tree?

29. Describe two common idioms in context-free grammars that cannot be parsed top-down.

30. What is the "dangling `else`" problem? How is it avoided in modern languages?

31. Discuss the similarities and differences between recursive descent and table-driven top-down parsing.

32. What are `FIRST` and `FOLLOW` sets? What are they used for?

33. Under what circumstances does a top-down parser predict the production $A \longrightarrow \alpha$

34. What sorts of "obvious" facts form the basis of `FIRST` and `FOLLOW` set construction?

35. Outline the algorithm used to complete the construction of `FIRST` and `FOLLOW` sets. How do we know when we are done?

36. How do we know when a grammar is not `LL(1)`?

37. What is the handle of a right sentential form?

38. Explain the significance of the characteristic finite-state machine in `LR` parsing.

39. What is the significance of the dot ($\bullet$) in an `LR` state?

40. What distinguishes the basis from the closure of an `LR` state?

41. What is a shift-reduce conflict? How is it resolved in the various kinds of `LR`-family parsers.

42. Outline the steps performed by the driver of a bottom-up parser.

43. What kind of parser is produced by `yacc/bison`? By `ANTLR`?

44. Why are there never any epsilon productions in an `LR(0)` grammar?

45. Why is syntax error recovery important?

46. What are cascading errors?

47. What is panic mode? What is its principal weakness?

48. What is the advantage of phase-level recovery over panic mode?

49. What is the immediate error detection problem, and how can it be addressed?

50. Describe two situations in which context-specific `FOLLOW` sets may be useful.

51. Outline Wirth's mechanism for error recovery in recursive descent parsers. Compare this mechanism to exception-based recovery.

52. What are error productions? Why might a parser that incorporates a high-quality, general-purpose error recovery algorithm still benefit from using such productions?

53. Outline the FMQ algorithm. In what sense is the algorithm optimal?

54. Why is error recovery more difficult in bottom-up parsers than it is in top-down parsers?

55. Describe the error recovery mechanism by `yacc/bison`.

56. What formal machine captures the behvaior of a scanner? A parser?

57. State three ways in which a real scanner differs from the formal machine.

58. What are the formal components of a DFA?

59. Outline the algorithm used to construct a regular expresion equivalent to a given DFA.

60. What is the inherent "big-O" complexity of parsing with a simulated NPDA? Why is this worse than the $O(n^3)$ time mentioned in Section 2.3?

61. How many states are there in an `LL(1)` PDA? Explain.

62. What are the variable prefixes of a CFG?

63. Summarize the proof that a DFA cannot recognize arbitrarily nested constructs.

64. Explain the difference between `LL` and `SLL` parsing.

65. Is every `LL(1)` grammar also `LR(1)`? Is it `LALR(1)`?

66. Does every `LR` language have an `SLR(1)` grammar?

67. Why are the containment relationships among grammar classes more complex than those among language classes?