1. true / false By a compiler we mean a program that translates to code that will run natively on some machine.

2. true / false ML can be compiled.

3. true / false FORTRAN can reasonably be considered the first programming language.

4. true / false Scheme and ML are eager (Sebesta calls them strict) functional languages and Haskell is a lazy functional language.

5. true / false The abstract data type for lists is predefined in Haskell.

6. true / false Functions can take tuples as an argument in Haskell.

7. true / false Type class constraints in Haskell appear before the =>.

8. true / false Programming on a Turing machine is very rudimentary.

9. true / false All local variables are local to some block.

10. true / false Haskell has an assignment, if, and while statement (but these are rarely used).

11. true / false There are tools in wide-spread use to generate parsers automatically from their descriptions.

12. true / false The function space type operator is right associative.

13. true / false If $P \Rightarrow Q$, then the set of computer states characterized by $Q$ is a subset of the set of state characterized by $P$.

14. true / false Constant expressions are not legal actual arguments for copy-out parameters.

15. true / false It is impossible to get a “no such method” error in Java (because Java is strongly typed).

16. true / false Prolog can be compiled.

17. true / false Functors have to be declared before they are used in Prolog.

18. true / false The syntax of a typical programming language can be expressed using regular expressions.

19. true / false Theoretically speaking, anything that can be programmed can be programmed in the Ada programming language.

20. true / false Lexical analysis determines the phrase structure of a language’s tokens.

21. true / false Pascal can be interpreted.
22. true / false If \( P \Rightarrow Q \), then the set of computer states characterized by \( Q \) is a subset of the set of state characterized by \( P \).

23. true / false Partial correctness assures termination of the program.

24. true / false Prolog builds a search space from the query, and then searches it.

25. true / false J is to APL as Beta is to SIMULA.

26. true / false The only practical algorithm for unification is \( O(n^2) \).

27. true / false HTML is a programming language.

28. true / false In ML \( \text{fn } x \Rightarrow x(2) \) is a higher-order function.

29. true / false Functional programming is slow because it is usually interpreted.

30. true / false The display is good only for languages with dynamic scoping.

31. true / false In a computer’s memory the bit patterns for objects allocated in the heap looks very much the same as the bit patterns for the program’s integer variables.

32. true / false Prolog uses the occurs check.

33. true / false A value whose type has an arrow in it is a function.

34. true / false There are tools in wide-spread use to generate scanners automatically from regular expressions.

35. true / false Functional programming is characterized by Cambridge prefix notation.

36. true / false Lists of functions are not possible in Haskell.

37. true / false Implicit coercion is an example of universal polymorphism.

38. true / false Parentheses are part of the syntax of function class in Haskell.

39. true / false Static scoping is the same as dynamic scoping for local variables.

40. true / false It makes no difference when the l-value of an actual arguments is computed for copy-in/copy-out parameters.

41. true / false Higher-order types are those having more than one “:” (cons) in them.

42. true / false A Prolog implementation requires garbage collection.

43. true / false A variable is an identifier.
44. true / false Logic investigates and classifies the structure of statements and arguments.

45. true / false Unification requires $O(N^2)$ running time.

46. true / false There exists a $O(n)$ algorithm for unification.

47. true / false A Prolog search space may have an infinite number of solutions.

48. true / false All functions in Haskell are higher-order.

49. true / false Arrow (function) types are, by their nature, contravariant in the domain and covariant in the range.

50. true / false Type names are capitalized in Haskell.

51. true / false “Branding” as in Modula-3 allows the programmer to make structural equivalence when the language supports name equivalence.

52. true / false A Haskell implementation will use a stack of activation records at runtime like Algol-like languages.

53. true / false Modula-3 uses name equivalence.

54. true / false In this class we will study the lambda calculus—the theory behind functional programming languages.

55. true / false Haskell is a language (one of the few) whose name is taken from the given or first name of a real person.

56. true / false Prolog has relations.

57. true / false It is not possible to read and write using standard IO in Haskell.

58. true / false Omitting the occurs-check makes Prolog logically sound.

59. true / false There is only one environment in which a procedure is defined.

60. true / false A formula of first-order logic can be used to characterize a set of computer states.

61. true / false Function application is right associative.

62. true / false The order of the rules in a Prolog program is a factor in determining the number of solutions in the search space.

63. true / false All recursion can be replaced by iteration.

64. true / false The stack is generally considered to be a very efficient way of storing local variables.
65. true / false Partial correctness means the program satisfies some of the postconditions.

66. true / false ALGOL uses call-by-name parameter passing.

67. true / false ALGOL can reasonably be considered the most influential programming language.

68. true / false It is possible to define a new function in Haskell without the programmer using formal parameters.

69. true / false There is only one environment in which a procedure is called.

70. true / false Variant records cannot be statically typed.

71. true / false ALGOL is an imperative language.

72. true / false In this class we will study the unification and resolution—the theory behind logic programming languages.

73. true / false A formal language is a set of symbols from an alphabet.

74. true / false Fortran, before FORTRAN 90, had no recursion and no pointers.

75. true / false In ML fn x => x(2) is an anonymous function.

76. true / false The unification problem has $O(n)$ time complexity.

77. true / false Frege contributed to the mathematical foundations of the theory of quantification.

78. true / false Function application is left associative.

79. true / false The array type operator in C# is covariant.

80. true / false All iteration can be replaced by recursion.

81. true / false Destructors are not commonly used in Haskell.

82. true / false The propositional formula $A \& B \rightarrow C$ is necessarily true if $B$ is false.

83. true / false Functors require heap allocation.

84. true / false The Hoare triple $\{P\} S \{false\}$ is valid.

85. true / false An ML function is a value whose type has an arrow in it.

86. true / false An Ada subtype gives the programmer a compile-time guarantee about the behavior of the program.

87. true / false C uses name equivalence.
88. true / false Ada uses name equivalence.
89. true / false An assertion is the same as a condition.
90. true / false Polymorphism means “many values.”
91. true / false Constructors can be used in the patterns used in function definitions.
92. true / false A different query gives rise to a different Prolog search space.
93. true / false Fortran has two different kinds of subprogram declarations: function and subroutine.
94. true / false The order in which Prolog clauses are asserted matters a great deal.
95. true / false An identifier is a variable.
96. true / false BNF describes more formal languages than does regular expressions.
97. true / false ML is an example of a non-procedural language.
98. true / false Localization of scope means assigning to variable exactly once.
99. true / false Structural equivalence is necessary to strengthen type abstraction.
100. true / false An ambiguous grammar and an unambiguous grammar may both describe the same language.
101. true / false Programming languages were created after computers where invented.
102. true / false “Sue me if my postcondition is false, sue you if my precondition is false.”
103. true / false A Haskell implementation will use a heap at runtime like Algol-like languages.
104. true / false All languages take basically the same approach to types.
105. true / false If a greedy quantifier matches some string, then the corresponding possessive quantifier will match too, although possibly matching fewer symbols.
106. true / false A package cannot be stateful in Ada.
107. true / false The Glasgow Haskell Compiler has an interactive interface.
108. true / false Haskell is lazy.
109. true / false   The types of high-order functions must have more than one
-> in them.

110. true / false   An Ada subtype is not a type.

111. true / false   Haskell can be interpreted.

112. true / false   Frege contributed to the mathematical foundations of the theory of quantification.

113. true / false   The Cherokee script is used in writing FORTRAN programs.

114. true / false   Modern languages tend to favor structural equivalence of types.

115. true / false   The most important data structure in functional languages is the immutable list.

116. true / false   Haskell is Turing complete.

117. true / false   Formal language theory applies to the lexical structure of programming languages, but not to the phrase structure.

118. true / false   Interactive language systems execute programs slower than compiled programs.

119. true / false   Prolog uses depth-first search (DFS) when searching for a solution.

120. true / false   Imperative programming is characterized by assignment, conditionals, and loops (gotos).

121. true / false   It is undecidable in general whether or not a variable will have the value zero during the execution of the program.

122. true / false   Prolog has functions.

123. true / false   It is impossible to automatically generate a program to do lexical analysis from a description of the tokens.

124. true / false   A Prolog query may have an infinite number of solutions, yet Prolog may find none of them.

125. true / false   The type operator for arrays in Java is covariant.

126. true / false   An interpreter never translates to intermediate code.

127. true / false   In a lazy language one can define infinite data structures directly.

128. true / false   Bounded quantification polymorphism is a combination of ad hoc and universal polymorphism.

129. true / false   The same sequence of bits can mean different things.
130. true / false Invisible things are not important.

131. true / false Lists are mutable data structures in Haskell.

132. true / false A type variable stands in place of a specific type.

133. true / false A program that cannot be statically type has a type insecurity.

134. true / false In an eager language one can define infinite data structures directly.

135. true / false The order in which Prolog clauses are asserted does not matter.

136. true / false All functions have names in Haskell.

137. true / false It is undecidable at compile time whether or nor a variable has a value in a specific range at run time.

138. true / false All formulas in first-order predicate logic can be represented by Prolog clauses.

139. true / false Backquotes make a identifier of a binary function parsed as infix in Haskell.

140. true / false Imperative programming languages differ from functional programming language in how they implement non-local variable access.

141. true / false Partial correctness assumes termination of the program.

142. true / false Prolog uses breadth-first search (BFS) when searching for a solution.

143. true / false Type variable is a lower-case letter in Haskell.

144. true / false Parentheses are part of the syntax of function calls in Haskell.

145. true / false Parametric universal polymorphism is the basis of object-oriented programming.

146. true / false A characteristic of universal polymorphism is a finite number of possibilities.

147. true / false Higher-order functions definitions—the bodies of functions—must have more than one \( \rightarrow \) in them.

148. true / false A significant weakness of denotational semantics is with concurrent programming.

149. true / false A loop invariant is a kind of logical assertion.
150. true / false  Prolog atoms are nullary functor symbols.
151. true / false  High-level programming languages have eliminated the need for debugging code.
152. true / false  Destructors are commonly used in Haskell.
153. true / false  Regular expressions are great because they are more expressive than other common formalisms.
154. true / false  Tuples of functions are possible in Haskell.
155. true / false  A package can be stateful in Ada.
156. true / false  With just two values a non-local variable reference can be found.
157. true / false  Functors are used to define functions in Prolog.
158. true / false  Functors are distinguished syntactically from predicates in Prolog.
159. true / false  Computer science is the study of computers.
160. true / false  A type insecurity arises when the data is misinterpreted.
161. true / false  In Haskell the index function (!!) is $O(1)$.
162. true / false  In Haskell \( \lambda x \Rightarrow x(2) \) is a higher-order function.
163. true / false  A canonical value is one which can be rewritten or simplified.
164. true / false  The programmer is required to declare the types of functions in Haskell.
165. true / false  Prolog relations can be defined recursively.
166. true / false  Universal, parametric polymorphism takes advantage of the fact that many programs don’t care about the specific types of its data.
167. true / false  Functions have one argument in Haskell.
168. true / false  “Sue me if my precondition is false, sue you if my postcondition is false.”
169. true / false  It is possible to statically type-check heterogeneous, composite data types with dynamic access.
170. true / false  Java uses name equivalence.
171. true / false  Prolog does a breadth-first search of the search space.
172. true / false In Prolog, a solution is found when all the goals have been established.

173. true / false Non-local variable access in a block-structured language can be implemented in a single machine instruction.

174. true / false An inference rule is a method of asserting the truth of one assertion on the basis on the form of other assertions.

175. true / false Ada uses call-by-name parameter passing.

176. true / false Prolog backtracks when no way can be found to make progress on the first goal.

177. true / false A loop invariant is a value that does not change.

178. true / false Cambridge prefix notation cannot be used to expression assignment or loops.

179. true / false Prolog is Turing complete.

180. true / false Java uses structural equivalences of types.

181. true / false Scheme is LISP cleaned up.

182. true / false Lists of functions are possible in Haskell.

183. true / false An interpreted program always executes slower than a compiled program.

184. true / false Like Haskell, Prolog uses arrays for complex data structures.

185. true / false All ML functions must be defined using fun or fn.