1. true / false A characteristic of universal polymorphism is an infinite number of possibilities.

2. true / false In ML \texttt{fn x => x(2)} is a higher-order function.

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

4. true / false Tuples of functions are possible in Haskell.

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

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

7. true / false Like most imperative languages, Scheme and OCaml evaluates function arguments before passing them to a function.

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

9. true / false Structural equivalence is necessary to strengthen type abstraction.

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

11. true / false Lists are immutable data structures in Haskell.

12. true / false Prolog relations can be defined recursively.

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

14. true / false All ML functions must be defined using \texttt{fun} or \texttt{fn}.

15. true / false Ada uses name equivalence.

16. true / false Fortran has two different kinds of subprogram declarations: \texttt{function} and \texttt{subroutine}.

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

18. true / false There are no implementations of Prolog.

19. true / false Programming languages were created after computers where invented.

20. true / false Referential transparency enables equational reasoning.

21. true / false Polymorphism means “many forms.”

22. true / false A function is not a canonical value. [hnf]

23. true / false Frege contributed to the mathematical foundations of the theory of quantification.
24. true / false  Higher-order functions definitions—the bodies of functions—must have more than one -> in them.

25. true / false  An Ada subtype is a type.

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

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

28. true / false  Prolog can be compiled.

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

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

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

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

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

34. true / false  Universal, parametric polymorphism takes advantage of the fact that many programs don’t care about the specific types of its data.

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

36. true / false  A variable is an identifier.

37. true / false  Haskell is lazy.

38. true / false  “Sue me if my postcondition is false, sue you if my precondition is false.”

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

40. true / false  In Prolog, a solution is found when all the goals have been established.

41. true / false  With just two values a non-local variable reference can be found.

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

43. 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 $P$. 
44. true / false  In ML $\texttt{fn } x \Rightarrow x(2)$ is an anonymous function.

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

46. true / false  Conjunction is denoted by a comma in Prolog.

47. true / false  A (function without side effects) is said to be strict if it is undefined (fails to terminate) when any of its arguments is undefined.

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

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

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

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

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

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

54. true / false  In Haskell every binary function name can be used in prefix form (function out front).

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

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

57. true / false  A subprocedure that takes another subprocedure as an argument is said to be higher-order.

58. true / false  Ada has inheritance as in OO languages using tagged record types.

59. true / false  HTML is a programming language.

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

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

62. true / false  ISO standard Prolog uses the occurs check.

63. true / false  It is possible to define a new function in Haskell without the programmer using formal parameters.
64. true / false It makes no difference when the l-value of an actual arguments is computed for copy-in/copy-out parameters.

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

66. true / false Visual Studio supports Haskell.

67. true / false Referential transparency is a property of language.

68. true / false BNF describes more formal languages than does regular expressions.

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

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

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

72. true / false Haskell is Turing complete.

73. true / false The order in which Prolog clauses are asserted matters a great deal.

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

75. true / false “Sue me if my precondition is false, sue you if my postcondition is false.”

76. true / false Localization of scope means assigning to variable exactly once.

77. true / false It is possible to statically type-check heterogeneous, composite data types with dynamic access.

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

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

80. true / false In Haskell backquotes make an identifier of a binary function parsed as infix.

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

82. true / false Partial correctness assures termination of the program. Partial correctness means the program satisfies some of the postconditions.

83. true / false Prolog does a breadth-first search of the search space.
true / false Theoretically speaking, anything that can be programmed can be programmed in the Ada programming language.

true / false Haskell is an example of a non-procedural language.

true / false Function arguments are separated by commas in Haskell.

true / false The implementation of a functional language, like Haskell or Scheme, will use a heap at runtime like Algol-like languages.

true / false Functors are distinguished syntactically from predicates in Prolog.

true / false A different query gives rise to a different Prolog search space.

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

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

true / false If a greedy quantifier matches some string, then the corresponding possessive quantifier will match too, although possibly matching fewer symbols.

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 \(P\).

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

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

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

true / false The programmer is required to declare the types of functions in Haskell.

true / false All ML functions must be defined using \texttt{fun} or \texttt{fn}.

true / false A loop invariant is a kind of logical assertion.

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

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

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

true / false The array type operator in C# is covariant.
104. true / false  It is impossible to get a “no such method” error in Java (because Java is strongly typed).
105. true / false  Functions have one argument in Haskell.
106. true / false  The only practical algorithm for unification is $O(n^2)$.
107. true / false  Type class constraints in Haskell appear before the $=>$.
108. true / false  Prolog is Turing complete.
109. true / false  Functional programming is characterized by Cambridge prefix notation.
110. true / false  Prolog has relations.
111. true / false  Haskell is a language (one of the few) whose name is taken from the given or first name of a real person.
112. true / false  The most important data structure in functional languages is the immutable list.
113. true / false  The propositional formula $A \& B \rightarrow C$ is necessarily true if $B$ is false.
114. true / false  Any binary file can be interpreted as a US-ASCII text file.
115. true / false  High-level programming languages have eliminated the need for debugging code.
116. true / false  Functors are used to define functions in Prolog.
117. true / false  A Prolog implementation requires garbage collection.
118. true / false  In Haskell the index function (!!) is $O(1)$.
119. true / false  Constructors can be used in Haskell function definitions as patterns.
120. true / false  An interpreted program always executes slower than a compiled program.
121. true / false  A package can be stateful in Ada.
122. true / false  The record \{a:int\} is a subtype of \{a:int, b:char\}.
123. true / false  A Haskell implementation will use a stack of activation records at runtime for non-local variable access like Algol-like languages.
124. true / false  The type operator for arrays in Java is contravariant.
125. true / false  All formulas in first-order predicate logic can be represented by Prolog clauses.
126. true / false Type names are capitalized in Haskell.
127. true / false The abstract data type for lists is predefined in Haskell.
128. true / false Implicit coercion is an example of universal polymorphism.
129. true / false Unification requires $O(n^2)$ running time.
130. true / false Type variables begin with a lower-case letter in Haskell.
131. true / false An interpreter never translates to intermediate code.
132. true / false All local variables are local to some block.
133. true / false Function application is right associative.
134. true / false There is only one environment in which a procedure is called.
135. true / false ML has an assignment, if, and while statement (but these are rarely used).
136. true / false Functions can take tuples as an argument in Haskell.
137. true / false The study of logic investigates empirical truth.
138. true / false C uses name equivalence.
139. true / false A loop invariant is a value that does not change.
140. true / false Static scoping is the same as dynamic scoping for local variables.
141. true / false Formal language theory applies to the lexical structure of programming languages, but not to the phrase structure.
142. true / false An assertion is the same as a condition.
143. true / false A value whose type has an arrow in it is a function.
144. true / false ML is an example of a non-procedural language.
145. true / false Destructors are commonly used in Haskell.
146. true / false Scheme is LISP cleaned up.
147. true / false It is undecidable in general whether or not a variable will have the value zero during the execution of the program.
148. true / false In Haskell $\lambda x \Rightarrow x(2)$ is a higher-order function.
149. true / false The types of high-order functions must have more than one $\rightarrow$ in them.
150. true / false In Haskell $\lambda x \Rightarrow x(2)$ is an anonymous function.
151. true / false  The same sequence of bits can mean different things.
152. true / false  There is only one environment in which a procedure is defined.
153. true / false  An ambiguous grammar and an unambiguous grammar may both describe the same language.
154. true / false  A canonical value is one which can be rewritten or simplified.
155. true / false  Modula-3 uses name equivalence.
156. true / false  All functions in Haskell are higher-order.
157. true / false  The Glasgow Haskell Compiler has an interactive interface.
158. true / false  Recursion in Prolog does require allocation of memory in the heap.
159. true / false  Interactive language systems execute programs slower than compiled programs.
160. true / false  A canonical value is one which can be rewritten or simplified.
161. true / false  An inference rule is a method of asserting the truth of one assertion on the basis on the form of other assertions.
162. true / false  Lexical analysis determines the structure of a language’s tokens.
163. true / false  Haskell can be interpreted.
164. true / false  Imperative programming is characterized by assignment, conditionals, and loops (gotos).
165. true / false  A formula of first-order logic can be used to characterize a set of computer states.
166. true / false  Pascal can be interpreted.
167. true / false  An identifier is a variable.
168. true / false  A type variable stands in place of a specific type.
169. true / false  All languages take basically the same approach to types.
170. true / false  Constant expressions are not legal actual arguments for copy-out parameters.
171. true / false  Invisible things are not important.
172. true / false  Ada uses call-by-name parameter passing.
173. true / false  Regular expressions are less expressive than other common ways of defining formal languages.
A type insecurity arises when the data is misinterpreted.

A literal can be both a fact and a query in Prolog.

In a lazy language one can define infinite data structures directly.

A subprocedure that takes another subprocedure as an argument is said to be higher-order.

The unification problem has $O(n)$ time complexity.

The function space type operator is right associative.

All iteration can be replaced by recursion.

Functors in Prolog do require allocation of memory in the heap.

ML can be compiled.

Prolog uses depth-first search (DFS) when searching for a solution.

Computer science is the study of computers.

Java uses name equivalence.

Lists of functions are possible in Haskell.

In Haskell every binary function name can be used in infix form (function in the middle).

Prolog atoms are nullary functor symbols.

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

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.

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

Turing machines are a good way to express computation.

Functional programming is slow because it is usually interpreted.

The display is good only for languages with dynamic scoping.
195. true / false The stack is generally considered to be a very efficient way of storing local variables.

196. true / false ALGOL is an imperative language.

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

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

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

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

201. true / false Prolog has construct specifically for functions.

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

203. true / false A formal language is a set of symbols from an alphabet.
1 Syntax

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

2. true / false Regular expressions are less expressive than other common ways of defining formal languages.

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

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

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

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

7. true / false An ambiguous grammar and an unambiguous grammar may both describe the same language.

8. true / false BNF describes more formal languages than does regular expressions.

9. true / false If a greedy quantifier matches some string, then the corresponding possessive quantifier will match too, although possibly matching fewer symbols.

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

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

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

2. true / false An assertion is the same as a condition.

3. 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 $P$.

4. true / false “Sue me if my precondition is false, sue you if my postcondition is false.”

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

6. true / false “Sue me if my precondition is false, sue you if my postcondition is false.”

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

1. true / false All languages take basically the same approach to types.

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

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

4. true / false C uses name equivalence.

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

6. true / false It is possible to statically type-check heterogeneous, composite data types with dynamic access.

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

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

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

10. true / false Java uses name equivalence.

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

12. true / false Structural equivalence is necessary to strengthen type abstraction.

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

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

15. true / false Polymorphism means “many values.”

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

17. true / false Ada has inheritance as in OO languages using tagged record types.

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

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

20. true / false The type operator for arrays in Java is contravariant.
21. true / false Arrow (function) types are, by their nature, contravariant in
the domain and covariant in the range.

22. true / false A type insecurity arises when the data is misinterpreted.

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

24. true / false Ada uses name equivalence.

25. true / false Any binary file can be interpreted as a US-ASCII text file.

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

27. true / false The same sequence of bits can mean different things.

28. true / false Universal, parametric polymorphism takes advantage of the
fact that many programs don’t care about the specific types
of its data.
1. true / false All formulas in first-order predicate logic can be represented by Prolog clauses.

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

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

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

5. true / false A literal can be both a fact and a query in Prolog.

6. true / false There are no implementations of Prolog.

7. true / false Conjunction is denoted by a comma in Prolog.

8. true / false Functors are distinguished syntactically from predicates in Prolog.

9. true / false Functors are used to define functions in Prolog.

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

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

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

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

14. true / false Prolog does a breadth-first search of the search space.

15. true / false It possible to define functions in (pure) Prolog.

16. true / false ISO standard Prolog uses the occurs check.

17. true / false Unification requires $O(n^2)$ running time.

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

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

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

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

22. true / false A different query gives rise to a different Prolog search space.

23. true / false Prolog atoms are nullary functor symbols.
24. true / false Prolog backtracks when no way can be found to make progress on the first goal.

25. true / false It is impossible to compile Prolog.

26. true / false Prolog has relations.

27. true / false Prolog is not Turing complete.

28. true / false Prolog relations can be defined recursively.

29. true / false In Prolog, a solution is found when all the goals have been established.

30. true / false The study of logic investigates empirical truth.

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

32. true / false Functors in Prolog do require allocation of memory in the heap.

33. true / false Recursion in Prolog does require allocation of memory in the heap.

34. true / false The order in which Prolog clauses are asserted does not matter.
5 Functional Programming

1. true / false In Haskell backquotes make an identifier of a binary function parsed as infix.

2. true / false In Haskell every binary function name can be used in infix form (function in the middle).

3. true / false In Haskell every binary function name can be used in prefix form (function out front).

4. true / false Function arguments are separated by commas in Haskell.

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

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

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

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

9. true / false A Haskell implementation will use a stack of activation records at runtime for non-local variable access like Algol-like languages.

10. true / false The implementation of a functional language, like Haskell or Scheme, will use a heap at runtime like Algol-like languages.

11. true / false A canonical value is one which can be rewritten or simplified.

12. true / false A subprocedure that takes another subprocedure as an argument is said to be higher-order.

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

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

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

16. true / false In Haskell the index function (!!) is $O(1)$.

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

18. true / false Haskell is an example of a non-procedural language.

19. true / false Parentheses are part of the syntax of function calls in Haskell.
20. true / false Destructors are commonly used in Haskell.

21. true / false Function application is right associative.

22. true / false The programmer is required to declare the types of functions in Haskell.

23. true / false Haskell can be interpreted.

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

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

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

27. true / false Type names are in lower-case in Haskell.

28. true / false Visual Studio supports Haskell.

29. true / false All functions are canonical values. [hnf]

30. true / false Constructors can be used in Haskell function definitions as patterns.

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

32. true / false Functions have one argument in Haskell.

33. true / false Haskell is Turing complete.

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

35. true / false Haskell is lazy.

36. true / false A canonical value is one which can be rewritten or simplified.

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

38. true / false A subprocedure that takes another subprocedure as an argument is said to be higher-order.

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

40. true / false In Haskell \( \lambda x \Rightarrow x(2) \) is a higher-order function.

41. true / false In Haskell \( \lambda x \Rightarrow x(2) \) is an anonymous function.

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

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

44. true / false Scheme is LISP cleaned up.
45. true / false  The Glasgow Haskell Compiler has an interactive interface.

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

47. true / false  The types of high-order functions must have more than one 
                 \( \rightarrow \) in them.

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

49. true / false  Like most imperative languages, Scheme and OCaml evaluates
                 function arguments before passing them to a function.

50. true / false  A (function without side effects) is said to be strict if it is
                 undefined (fails to terminate) when any of its arguments is
                 undefined.

51. true / false  Tuples of functions are possible in Haskell.

52. true / false  Type class constraints in Haskell appear before the \( \rightarrow \).

53. true / false  Type variables begin with a lower-case letter in Haskell.