FINAL EXAM: CS3342b Tuesday, 25 April 2017, 2pm, Room FEB GYM

NAME AS APPEARS ON STUDENT ID:

STUDENT ID NUMBER:

GAUL/CONFLUENCE USER NAME:

REMINDERS (from course outline):

1. The final exam will be closed book, closed notes, with no electronic devices allowed, with particular reference to any electronic devices that are capable of communication and/or storing information.

1. Adding a feature to a programming language to make it easier to do something that was already doable is called adding ANSWER.
   - syntactic sugar
2. Matz, the creator of Ruby, thinks that it is less important to optimize the execution (efficiency) of a programming language and more important to optimize the efficiency of ANSWER.
   - the programmers
3. A programming language is called ANSWER if it is executed directly by an interpreter rather than by first being compiled with a compiler.
   - interpreted
4. If the types of a programming language are bound at execution time rather than compile time, then the types are called ANSWER.
   - dynamically typed
5. In describing the properties of an object-oriented language, encapsulation means ANSWER.
   - data and behavior are packaged together
   - there is a mechanism for restricting access to an object's components
6. In discussing object-oriented languages, objects are organized into a class tree to support the property of ANSWER.
   - inheritance
7. In discussing object-oriented languages, being able to handle objects of related types is called ANSWER.
   - polymorphism
8. The application that caused a significant increase in the popularity of Ruby was a web framework called ANSWER.
   - Rails
   - Ruby on Rails
9. The main concurrency approach used in Ruby is ANSWER.
   - threads
10. The command name for the Ruby interpreter is ANSWER.
    - irb
11. In Ruby, true.class returns ANSWER.
    - TrueClass
12. Ruby supports two common ways that boolean expressions are handled in programming languages. In one approach both subexpressions of a boolean operator are evaluated before the boolean operator is evaluated. In the other approach, called ANSWER, the first subexpression in a boolean expression is evaluated and if that is enough to know the result of the boolean expression, then the second subexpression is not evaluated.
    - short-circuit evaluation

13. In Ruby, normally, when you try to add a String to a Fixnum, you get an error message saying that a String can't be coerced to a Fixnum. This is because Ruby is ANSWER typed.

    - strongly

14. One way of checking types is to see what constructor was used to create an object that is a parameter. Another way of checking types is to wait until a method is sent to an object and see if it supports the method. This second way is called ANSWER

    - duck typing

15. A major claim in object-oriented design philosophy is that you should code to ANSWER rather than code to implementation.

    - interface

16. The & notation in the line of Ruby def george(&sam) is used to indicate that sam is ANSWER.

    - a code block

17. The : notation in the Ruby expressions :hi is used to indicate that hi is ANSWER.

    - a symbol

18. With respect to the value returned by the Ruby expression 'hi'.object_id == 'hi'.object_id, you can say it ANSWER.

    - could be either true or false

19. With respect to the value returned by the Ruby expression :hi.object_id == :hi.object_id, you can say it ANSWER.

    - will always be true

20. To execute a code block in Ruby that is passed to a method but doesn't appear on its parameter list, you use the keyword ANSWER.

    - yield

21. To execute a code block in Ruby that is passed to a method on its parameter list, you send that parameter the method ANSWER.

    - call

22. A code block in Ruby is some lines of code surrounded by either curly braces or ANSWER.

    - do end

23. In Ruby, the expression Fixnum.class returns ANSWER.

    - Class

24. The root of the inheritance hierarchy in Ruby is the class ANSWER.

    - Object

25. In Ruby, the name of the method in the class Me that is automatically invoked when a new object of type Me is created with Me.new is ANSWER.

    - initialize

26. In Ruby, the @ is used to indicate that the variable @me is ANSWER.

- an instance variable

27. In Ruby, the @@ is used to indicate that the variable @@me is ANSWER.

    - a class variable

28. In Ruby, by convention, the ? in the method me? is used to indicate that me is ANSWER.

    - boolean

29. In Ruby, the mixin is used to solve the object-oriented programming problem of ANSWER.

    - multiple inheritance

30. The feature of programs being able to 'write programs' (creating application specific language features) is called ANSWER.

    - metaprogramming

31. In Ruby, if you declare a class with a class name that is already in use and put in it the definition of a new method, you have changed the functionality of the existing class (even if it is a predefined class like Fixnum). The property of Ruby that allows this is ANSWER.

    - open classes

32. When you send a message to a Ruby object, Ruby first looks at the methods that object supports, and then starts working the inheritance chain. If it still can't find the appropriate method, the message and its parameters get passed as a message to the object looking for a method called ANSWER.

    - method_missing

33. In the Ruby community, the acronym DSL is an abbreviation for ANSWER.

    - domain specific language

34. In Ruby, if a line starts with a method name, that method is being sent to the object named ANSWER.

    - self

35. When you define a method in a class, normally it is meant to be invoked on an object of that class (an instance method). Sometimes it is meant to be invoked on the class name itself (a class method), like Date.parse('3rd Feb 2001'). In Ruby, to define a class method, we put ANSWER at the beginning of the method name in its definition.

    - self.

36. Instead of the + symbol, Haskell uses the symbol ANSWER for a string concatenation operator.

    - ++

37. The type of a string constant in Haskell, by default, is written ANSWER.

    - [Char]

38. In Haskell, you use the keyword ANSWER to collect related code into a similar scope.

    - module

39. In Haskell, if I define a function double = x + x, its type signature would be ANSWER.

    - $(Num \quad a) \Rightarrow a \rightarrow a$

40. In Haskell, instead of writing something like if x == 0 then 1 else fact (x - 1) * x, you can write a series of lines starting with factorial 0 = 1. This second style is called ANSWER.

    - pattern matching

41. In Haskell, instead of writing something like if x == 0 then 1 else fact (x - 1) * x, you can write a series of lines starting with | x ¿ 1 = x * factorial (x - a). This second style is called ANSWER.

    - using guards

42. In Haskell, instead of writing something like second x = head(tail(x)), you can write this without introducing the parameter x by using function composition. Doing that, you would write ANSWER.

    - second = head . tail

43. In Haskell, if I write (h:t) = [3, 5, 7], ANSWER is the value of h.

    - 3

44. In Haskell, if I write (h:t) = [3, 5, 7], ANSWER is the value of t.

    - [5, 7]

45. In Haskell, ANSWER is the output of zip [17..20] [10,8..4].

    - [(17,10),(18,8),(19,6),(20,4)]

46. In Haskell, ANSWER is the output of zip [20..17][10, 8..4].

    - []

47. In Haskell, defining lists using a notation like $[x * 2 | x \leftarrow [3, 4, 5]]$ is called using ANSWER.

    - list comprehensions

48. In Haskell, $[x * 2 | x \leftarrow [3, 4, 5]]$ evaluates to ANSWER.

    - $[6, 8, 10]$

49. In Haskell, how would you write an anonymous function so that the expression 'map ANSWER $[1, 2, 3]$' produces $[-4, -5, -6]$.

    - $(x \rightarrow -(x + 3))$

50. In Haskell, if we want to define a local named function inside a function definition, we use the keyword ANSWER.

    - where

51. In Haskell, the type signature of the function sum x y = x + y is ANSWER.

    - $(Num \quad a) => a \rightarrow a \rightarrow a$

52. In Haskell, given the definition sum x y = x + y, ANSWER is the value of that is produced by the expression (sum 3).

    - $(x \rightarrow 3 + x)$

53. The way Haskell handles functions with more than one parameter is called ANSWER.

    - currying

54. In most languages, a function definition like f a b = a : (f (a + b) b) would result in an infinite recursion. However, in Haskell we can partially evaluate functions like this because Haskell is based on ANSWER.

    - lazy evaluation

55. Although Haskell is a statically typed language, we usually don't need to write type declarations because Haskell uses ANSWER to figure out what the types are.

    - type inference

56. In Haskell, we can declare the type of a parameter to a function to be something specific like Char. However, we can also declare the type of a parameter to be something that could include many types like ListLike that supports the functions head and tail. We do this with a definition of ListLike that begins with the keyword ANSWER.

    - class

57. One of the three most significant parts of a monad is called ANSWER, which wraps up a function and puts it in the monad's container.

    - return

58. One of the three most significant parts of a Haskell monad is called ANSWER, which unwraps a function.

    - >>=
    - a bind function

59. In Haskell's do notation for working with monads, assignment uses the ANSWER operator.

    - ←

60. Since Haskell doesn't have traditional error handling, by convention, people use the ANSWER monad to distinguish a valid return from an error return.

    - Maybe

61. When viewing programming languages as natural languages, the word ANSWER is used instead of 'words'.

    - tokens

62. The routine in a compiler that takes as input a sequence of characters outputs these characters grouped into meaningful units is called ANSWER.

    - a lexical analyzer
    - a scanner
    - a lexer

63. The specifications for how to group characters into meaningful units are traditionally written as ANSWER.

    - regular expressions

64. The specifications of how to group characters into meaningful basic units of a programming language are generlly implemented in code that has the abstract form of ANSWER.

    - a finite automata

- a finite state machine

65. When viewed formally, a language is defined as a set of ANSWER.

    - strings

66. The Greek letter epsilon, when talking about languages, is used to represent ANSWER.

    - the empty string

67. In automatically generating the code that reads characters and outputs the part of a programming language that is analogous to its words, we start with a specification and then traditionally convert it into code in two stages. In the first stage, we produce ANSWER.

    - a nondeterministic finite automata
    - a nondeterministic finite state machine

68. In automatically generating the code that reads characters and outputs the part of a programming language that is analogous to its words, we start with a specification and then traditionally convert it into code in two stages. The main problem that can arise in moving from the first stage to the second stage is ANSWER.

    - an exponential explosion in the number of states needed

69. Three concepts related to concurrency were discussed with regards to the language Io. ANSWER was presented as a way to manage two execution streams that pass control back and forth between themselves.

    - coroutines

70. Three concepts related to concurrency were discussed with regards to the language Io. ANSWER was presented as a general mechanism for sending a message to an object that would cause that object to respond to the message as a separate process running asynchronously.

    - Actors

71. Three concepts related to concurrency were discussed with regards to the language Io. ANSWER was presented as a way to request that something be computed and then be able to continue computing until the result was needed. If the result was available then things would proceed as expected. If the result was not available, then a wait would be initiated until the result became available.

    - Futures

72. Io is known for taking ANSWER -based approach to object-oriented programming.

    - a prototype

73. In Io, the basic method for creating a new object is ANSWER.

    - clone

74. In Io, the type of an object is generally the nearest ancestor that ANSWER.

    - has a name that starts with a capital letter
    - has a slot for the method type

75. In Io, we create a singleton by redefining the method ANSWER.

    - clone

76. In Ruby, the evaluation of arguments to a message are handled by the object sending the message. In Haskell, the runtime environment decides when and how much to evaluate an argument to a function. In Io, the evaluation of the arguments to a message is made by ANSWER.

    - the reciever of the message

77. In Io, a message has three aspects that can be interrogated by the call method. They are: the sender, the reciever, and ANSWER.

    - the argument list

78. Io allows programmers to play with its syntax, doing things like introducing a colon operator and redefining how curly braces are processed. This makes it easy to use Io to create ANSWER.

    - Domain Specific Languages
    - DSLs

79. As one would expect in an object-oriented language, when a message is sent to an object, the first thing the system does is to look for a corresponding method in that object. However, Io lets you change what happens next by redefining the method named ANSWER.

    - forward

80. The central idea of context-free grammars is to define a language by productions. These productions say that a nonterminal symbol can be replaced by ANSWER.

    - a sequence of terminals and nonterminals
    - a sequence of symbols

81. The specific type of context-free grammar that was the main focus of the portion of the Syntax Analysis chapter that was assigned was ANSWER.

    - LL(1)

82. In a context-free grammar, the nonterminal that derives an entire member of the language being defined is called ANSWER.

    - a start symbol

83. Using the context-free grammar based on the two rules $A \to bA$ and $A \to b$, ANSWER would be the derivation sequence for bbb.

    - $A \Rightarrow Ab \Rightarrow Abb \Rightarrow bbb$

84. ANSWER is the regular expression that corresponds to the language defined by the context-free grammar with the three rules $A \to Aa$, $A \to Ab$, $A \to a$.

    - $a(a|b)*$

85. ANSWER would be the derivation of ((1)) in the language defined by the context-free grammar consisting of the two rules $E \to (E)$ and $E \to 1$.

    - $E \Rightarrow (E) \Rightarrow ((E)) \Rightarrow ((1))$

86. ANSWER are two derivations of the string cc that produce distinct syntax trees from the context-free grammar $X \to XcY$, $Y \to X$, $Y \to$ and $X \to$.

    - $X \Rightarrow XcY \Rightarrow XcYcY \Rightarrow cYcY \Rightarrow ccY \Rightarrow cc$ AND $X \Rightarrow XcY \Rightarrow XcX \Rightarrow XcXcY \Rightarrow cXcY \Rightarrow ccY \Rightarrow cc$

87. When a grammar can produce two distinct syntax trees for the same string, the grammar is said to be ANSWER.

    - ambiguous

88. If I wanted to fix the grammar $E \rightarrow E + E$ and $E \rightarrow id$, so that it would only produce one syntax tree, which is left recursive; the new grammar would be ANSWER.

    - $E \rightarrow E + F$ and $E \rightarrow F$ and $F \rightarrow id$

89. One aspect of the if-then-else-end syntax of Ruby is that it avoids the ANSWER problem.

    - dangling else

90. In the context-free grammar $A \rightarrow BA$, $B \rightarrow AB$, $A \rightarrow B$, $A \rightarrow a$, $B \rightarrow b$, and $B \rightarrow$ the value of Nullable(A) is ANSWER.

    - true

91. In the context-free grammar $A \rightarrow BA$, $B \rightarrow AB$, $A \rightarrow a$, $B \rightarrow b$, $B \rightarrow$ the value of Nullable(A) is ANSWER.

    - false

92. In the context-free grammar $A \rightarrow BA$, $B \rightarrow AB$, $A \rightarrow B$, $A \rightarrow a$, $B \rightarrow b$, and $B \rightarrow$ the value of FIRST(A) is ANSWER.

    - a, b

93. In the context-free grammar $A \rightarrow BA$, $B \rightarrow AB$, $A \rightarrow a$, $B \rightarrow b$, $B \rightarrow$ the value of FIRST(A) is ANSWER.

    - a, b

94. In the context-free grammar $A \rightarrow BA$, $B \rightarrow AB$, $A \rightarrow B$, $A \rightarrow a$, $B \rightarrow b$, and $B \rightarrow$ the value of FOLLOW(A) is ANSWER.

    - a, b

95. In the context-free grammar $A \rightarrow BA$, $B \rightarrow AB$, $A \rightarrow a$, $B \rightarrow b$, $B \rightarrow$ the value of FOLLOW(A) is ANSWER.

    - b

96. The context-free grammar $A \rightarrow BA$, $B \rightarrow AB$, $A \rightarrow a$, $B \rightarrow b$, $B \rightarrow$ is not LL(1) specifically because ANSWER.

    - FIRST(BA) and FIRST(a) both include a, so we do not know which A rule to use

97. When you write a parser for a context-free grammar that satisfies the LL(1) criteria by representing each non-terminal by a function that chooses what functions to invoke by the LL(1) criteria, this sort of parser is called ANSWER.

    - a recursive descent parser

98. Programming languages that view programming as describing a step-by-step process to do something are called ANSWER languages.

    - imperative

99. Programming languages that view programming as describing characteristics of the problem domain and characteristics of the solution and leaving it to the language processor to find a solution are called ANSWER languages.

    - declarative

100. In Prolog, the most natural way to express the fact that 'a lion is a cat' is ANSWER.

    - cat(lion).
    - is_a(lion, cat).

101. In Prolog, the most natural way to express the query 'what animals are cats?' is ANSWER.

    - cat(What).
    - cats(What).
    - is_a(What, cat).
    - are(What, cats).

102. In Prolog, the most natural way to express the rule that 'I am an ancestor of you if I am a parent of you' is ANSWER.

    - ancestor(I, You) :- parent(I, You).

103. In Prolog, the most natural way to express the rule that 'I am an ancestor of you if I am a parent of an ancestor of you' is ANSWER.

    - ancestor(I, You) :- parent(I, Ancestor), ancestor(Ancestor, You).

104. In Prolog, the expression hi(X, 4) = hi(3, Y) causes X to have the value ANSWER.

    - 3

105. In Prolog, the expression hi(X, 4) = hi(3, Y) causes Y to have the value ANSWER.

    - 4

106. In Prolog, the expression hi(X, 4) = hi(3, X) causes X to have the value ANSWER.

    - X will not be bound and the expression will fail
    - X will not be bound

107. In Prolog, the expression $[1, 2, 3] = [X|Y]$ causes X to have the value ANSWER.

    - 1

108. In Prolog, the expression $[1, 2, 3] = [X|Y]$ causes Y to have the value ANSWER.

    - $[2, 3]$

109. In Prolog, the expression $X = [[1, 2]|[3, 4]]$ causes X to have the value ANSWER.

    - $[[1, 2], 3, 4]$

110. In Prolog, the expression X = 1 + 2 causes X to have the value ANSWER.

    - 1+2

111. In Prolog, the expression 2 = 1 + X causes X to have the value ANSWER.

    - X remains unbound

- X remains unbound and the expression fails

112. In Prolog, the expression that would cause an unbound variable X to take on the sum of the values of a bound variable Y and a bound variable Z is ANSWER.

    - X is Y + Z

113. Each named object will have ANSWER, where the name is defined as a synonym for the object.

    - a declaration

114. The technical term for connecting a name with an object is ANSWER.

    - binding

115. The portion of the program where the name is visible is called its ANSWER.

    - scope

116. When the structure of the syntax tree is used to determine which object corresponds to a name, this is called ANSWER.

    - static scoping
    - lexical scoping

117. A compiler typically keeps track of which names are associated with which objects by using ANSWER.

    - a symbol table
    - an environment

118. ANSWER data structures have the property that no operation on the structure will destroy or modify it.

    - persistent
    - functional
    - immutable

119. ANSWER data structures have the property that there are operations on the structure can destroy or modify it.

    - imperative
    - destructively updated
    - mutable

120. Since a compiler may have to look up what object is associated with a name many times, it is typical to use ANSWER to avoid linear search times.

    - hash tables

121. In the ICD textbook's example interpreter for evaluating expressions, in the row labelled id, we have the code: v = lookup(vtable, getname(id)) ; if v = unbound then error() else v. It says getname(id) instead of id, because ANSWER.

    - id indicates a token with a type and value field

122. In the ICD textbook's example interpreter for evaluating expressions, in the row labelled id, we have the code: v = lookup(vtable, getname(id)) ; if v = unbound then error() else v. The value of v would be unbound in the situation that ANSWER.

- getname(id) was not declared
- getname(id) was not bound

123. In the ICD textbook's example interpreter for evaluating expressions, in the row labelled id(Exps), we have the code: args = EvalExps(Exps,vtable,ftable). We pass vtable to EvalExps to handle ANSWER.

    - expressions that contain identifiers

124. In the ICD textbook's example interpreter for evaluating expressions, in the row labelled id(Exps), we have the code: args = EvalExps(Exps,vtable,ftable). We pass ftable to EvalExps to handle ANSWER.

    - expressions that contain function usages

125. In the ICD textbook's example interpreter for evaluating expressions, in the row labelled let id = Exp1 in Exp2, we have the code: v1 = EvalExp(Exp1, vtable, ftable); vtableP = bind(vtable, getname(id), v1), EvalExp(Exp2, vtableP, ftable). The bind function changes vtable into vtableP by ANSWER.

    - inserting the association of getname(id) with the value v1 into the table
    - inserting the binding of getname(id) with the value v1 into the table

126. Scala was designed to connect two programming paradigms, which were ANSWER.

    - object-oriented and functional

127. Another design goal for Scala was to have its programs easily interoperate with those written in ANSWER.

    - Java

128. Scala is ANSWER typed

    - statically

129. Scala uses few type declarations because its compiler does ANSWER.

    - type inferencing

130. The main concurrency method used in Scala is ANSWER.

    - actors

131. In Scala, to indicate that a variable is immutable, you introduce it with the ANSWER keyword.

    - val

132. In Scala, to indicate that a variable is mutable, you introduce it with the ANSWER keyword.

    - var

133. In Scala, if I want to redefine a method that is defined in my parent class, I indicate this by using the keyword ANSWER.

    - override

134. The Scala feature closest to a Ruby mixin is the ANSWER.

    - trait

135. In Scala, the type that every type is a subtype of is called ANSWER.

    - Any

136. In Scala, the type that is a subtype of every type is called ANSWER.

    - Nothing

137. Many programming languages represent internal constants for types like strings, floats, and integers. Scala has the unusual distinction of having an internal constant representation for the type ANSWER, which is normally viewed as a format external to a program.

    - XML

138. The ! in Scala is used to ANSWER.

    - send a message to an actor

139. In the chapter on Scala, we get the following interesting quote: ANSWER is the most important thing you can do to improve code design for concurrency.

    - Immutability

140. In Erlang, the main approach to concurrency is ANSWER.

    - actors

141. In the Erlang community, ANSWER code refers to replacing pieces of your application without stopping your application.

    - hot-swapping

142. An unusual built-in constant construct in Erlang lets us write ¡¡4:3,1:3¿¿ to represent the value ANSWER.

    - !
    - octal 41
    - decimal 33
    - hexidecimal 21

143. Many syntax features of Erlang, such as ending statements with a period, reflect the influence of the programming language ANSWER.

    - Prolog

144. In Erlang, you can link two processes together. Then when one dies, it sends ANSWER to its twin.

    - an exit signal

145. The main programming paradigm in Erlang is ANSWER programming.

    - functional

146. In Ruby, you would group methods into a class. In Erlang, you group functions into ANSWER.

    - a module

147. The idea that when a process has an error, it is up to a monitoring process to determine what to do about the problem is referred to by the motto ANSWER in Erlang.

    - Let It Crash

148. Unlike most Lisp systems, Clojure doesn't use its own custom virtual machine. It was originally designed to compile to code that would run on the ANSWER.

- JVM
- Java Virtual Machine

149. The main programming paradigm for Clojure is ANSWER programming.
    - functional

150. The loop and recur constructs are in Clojure to guide ANSWER.
    - tail recursion optimization

151. In Clojure, the value of (repeat 1) is ANSWER.
    - an infinite sequence of 1s
    - a lazy infinite sequence of 1s

152. In Clojure, (take 3 (iterate (fn [x] (* 2 x)) 2)) produces ANSWER.
    - (4 8 16)

153. The main Clojure approach to concurrency is called ANSWER.
    - Software Transactional Memory
    - STM

154. In Clojure, ANSWER is a concurrency construct that allows an asynchronous return before computation is complete.
    - a future

155. In Clojure, you cannot change a reference outside of ANSWER.
    - a transaction

156. One approach to speeding up an interpreter is to translate pieces of the code being interpreted directly into machine code during program execution, this is called ANSWER.
    - just-in-time compilation

157. The technical term for the compiler design methodology where the translation closely follows the syntax of the language is ANSWER.
    - syntax-directed translation

158. Using the straightfoward expression translation scheme in the textbook, if I were to TransExp('3 * x + 1', vtable, ftable), newvar() will be invoked ANSWER times.
    - 5

159. Using the straightfoward statement translation scheme in the textbook, if I were to TransStat('if true then z := 1 else z := 2', vtable, ftable), newlabel() be invoked ANSWER times.
    - 3

160. Using the straightfoward statement translation scheme in the textbook, if I were to TransStat('while true do z := 1 + z', vtable, ftable), newlabel() be invoked ANSWER times.
    - 3

161. Using the straightfoward statement translation scheme in the textbook, if I were to TransStat('while z < 3 do z := 1 + z', vtable, ftable), newvar() be invoked ANSWER times.

- 5

162. Type checking done during program execution is called ANSWER.

    - dynamic typing

163. Type checking done during program compilation is called ANSWER.

    - static typing

164. ANSWER typing is when the language implementation ensures that the arguments of an operation are of the type the operation is defined for.

    - Strong

165. ANSWER is the data structure used in language translation to track the binding of variables and functions to their type.

    - A symbol table

166. The different traversals of a syntax tree done during compilation associate information with the nodes of the tree. The technical term for this kind of information is ANSWER.

    - attributes

167. ANSWER means that the language allows the same name to be used for different operations over different types.

    - Overloading

168. Some languages allow a function to be ANSWER, that is to be defined over a large class of similar types, e.g., over arrays no matter what type their elements are.

    - polymorphic
    - generic

169. When a function is invoked, if the language passes a copy of the value of each parameter to the code that performs the function, this is called ANSWER.

    - call-by-value

170. If the system stack is used for a call stack, then it becomes important for the caller to update the top of the stack before copying items into it. The reason is because we are worried about the top of the stack being changed by ANSWER after we have copied in information but before we updated the stack top.

    - an interrupt

171. The portion of the call stack associated with a single function invocation and running is called ANSWER.

    - an activation record

172. Another method of parameter passing, whose technical name is ANSWER, is implemented by passing the address of the variable (or whatever the given parameter is). Assigning to such a parameter would then change the value stored at the address.

    - call-by-reference

173. In C, when you pass a function as a parameter to another function, it is implemented as passing ANSWER.

    - the address of the start of the function code