Study Questions
Programming Languages

December 18, 2017

1. Fill in the blanks with the correct programming language from the following list:

   Ada, ALGOL, ALGOL-W, APL, BASIC, Beta, C, C++, C#, CLU, COBOL, Eiffel, Forth, FORTRAN, Icon, Haskell, J, Java, LISP, ML, Mesa, Modula-2, Modula-3, Oberon, Pascal, PL/I, PROLOG, Python, Simula, SmallTalk, SNOBOL

   (a) Back in the 1960's, the language that introduced the notion of class and object was $\boxed{}$. Its modern successor is $\boxed{}$.

   (b) Surprisingly the programming language $\boxed{}$ is used today in financial applications. It was originally designed in the 1960's to describe hardware and relies heavily on arrays and vectors. It has a modern successor called $\boxed{}$.

   (c) Kenneth Iverson won the Turing Award (1979) for designing $\boxed{}$ while working at IBM around 1960.

   (d) Like FORTRAN, $\boxed{}$ was developed as an IBM product. It is an early attempt to design a language for all application areas, and includes concurrently executing tasks, exceptions, and pointers.

   (e) The first high-level programming language was $\boxed{}$.

   (f) Betrand Meyer developed the $\boxed{}$ programming language which is not type-safe because it violates the law of contravariance.

   (g) Grace Hopper led the development of $\boxed{}$, a programming language for business applications.

   (h) The most influential programming language is $\boxed{}$.

   (i) Popular in astronomic circles, the $\boxed{}$ programming language is based on a stack.

   (j) The latest programming language designed by Niklaus Wirth is $\boxed{}$, which is both strongly-typed and object-oriented.

   (k) $\boxed{}$ was defined at Xerox Palo Alto Research Center in the late 1970's for systems programming.

   (l) $\boxed{}$ was jointly designed by the Systems Research Center of Digital Equipment Corporation in Palo Alto and the Olivetti Research Center in Menlo Park (neither exist anymore). It is based on Modula-2, Mesa, and Cedar, and it is very similar to Oberon.

   (m) An early string-oriented programming language, $\boxed{}$, was developed in the early 1960s by Ralph Griswold at Bell Labs. Its Pascal-like successor is $\boxed{}$.

   (n) Niklaus Wirth and C. A. R. Hoare designed the language that is known as

$\boxed{}$ in response to the direction that the ALGOL 68 development was taking.

(o) $\boxed{}$ was designed at Dartmouth College specifically for instructional use, but has since seen widespread use on micro-processors.

(p) Niklaus Wirth added modules to Pascal in creating $\boxed{}$.

(q) $\boxed{}$ is the quintessential logic programming language.

(r) $\boxed{}$ was the first language to use a recursive data structure, the list.

(s) Both successors to C++, $\boxed{}$, by Sun Microsystems, and $\boxed{}$, by rival MicroSoft, are very similar.

(t) In $\boxed{}$ programs and data have the same form.

(u) $\boxed{}$ is a strongly-typed functional programming language with exception handling and modules that does not use lazy evaluation.

2. Fill in the box with the number of the most appropriate choice from the left.

(a) $\boxed{}$ LISP		(1) strings
(b) $\boxed{}$ APL		(2) artificial intelligence
(c) $\boxed{}$ SNOBOL		(3) scientific, engineering
(d) $\boxed{}$ COBOL		(4) scalars, vectors, matrices
(e) $\boxed{}$ FORTRAN		(5) liberal arts majors
(f) $\boxed{}$ BASIC		(6) graphics
		(7) recursive lists
		(8) business
		(9) embedded systems

3. Fill in the box with the number of the most appropriate choice from the left.

(a)	☐	Ada	
(b)	☐	ALGOL	
(c)	☐	APL	
(d)	☐	BASIC	
(e)	☐	Beta	
(f)	☐	COBOL	
(g)	☐	C++	
(h)	☐	C	
(i)	☐	FORTRAN	
(j)	☐	Icon	
(k)	☐	Java	
(l)	☐	LISP	
(m)	☐	ML	
(n)	☐	Pascal	
(o)	☐	Perl	
(p)	☐	Python	
(q)	☐	Simula	
(r)	☐	SNOBOL	

1 Augusta Ada Bryon
2 John Backus
3 Dahl and Nygaard
4 Ralph Griswold
5 Kenneth Iverson
6 James Gosling
7 Kemeny and Kurtz
8 Guido van Rossum
9 John McCarthy
10 Robin Milner
11 Kernighan and Richie
12 Bjarne Stroustrup
13 US DoD, G. Hopper
14 US DoD, J. Icbaih
15 Larry Wall
16 IBM

4. Fill in the box with the number of the most appropriate choice from the left.

(a)	☐	John Backus	(j)	☐	Grace M. Hopper
(b)	☐	Friedrich L. Bauer	(k)	☐	Robert Kowalski
(c)	☐	Lewis Carroll	(l)	☐	Donald E. Knuth
(d)	☐	Noam Chomsky	(m)	☐	John McCarthy
(e)	☐	E. W. Dijkstra	(n)	☐	Robin Milner
(f)	☐	Gottlob Frege	(o)	☐	David Parnas
(g)	☐	James Gosling	(p)	☐	Dennis Ritchie
(h)	☐	David Gries	(q)	☐	Bertrand Russell
(i)	☐	C. A. R. Hoare	(r)	☐	Dana S. Scott

(s)	$\boxed{}$	Bjarne Stroustrup
(t)	$\boxed{}$	Alan Turing
(u)	$\boxed{}$	Larry Wall
(v)	$\boxed{}$	Nikolas Wirth
(w)	$\boxed{}$	Konrad Zuse

(1) German member of ALGOL design team

(2) ML, Turing Award 1991

(3) known for axiomatic semantics

(4) pseudonym for Charles Dodgson

(5) author of "Jabberwocky"

(6) co-author of *The Practice of Prog.*

(7) author of *A Discipline of Programming*

(8) author of *The Science of Programming*

(9) author of *The Art of Computer Prog.*

(10) 19th century German logician

(11) Designer of C, developer of UNIX

(12) Turing Award 1983

(13) Univ of Edinburgh, PROLOG

(14) MIT linguist

(15) famous Vietnam war critic,

(16) 20th century British mathematician

(17) led FORTRAN design team

(18) BNF named for him

(19) influenced COBOL

(20) LISP and ALGOL designer

(21) inventor of C++

(22) inventor of Java

(23) PERL

(24) associated with modularity

(25) inventor of Modula-2, and Oberon

(26) Turing award 1984

(27) decrypter of WW II Enigma

(28) developed an early computer

5. Fill in the box on the left with the number of the most appropriate choice from the right.

(a)		referential transparency
(b)		semantics
(c)		declarative
(d)		notation
(e)		orthogonality
(f)		abstraction
(g)		scientific method
(h)		visualization
(i)		expression
(j)		Sapir-Whorf hypothesis
(k)		denotation
(l)		paradigm
(m)		model
(n)		language
(o)		pictograph

1 signification of words or forms
2 the medium of expression
3 language confines thought
4 without specific instances
5 what versus how
6 a system or set of marks, signs, or figures
7 graphic presentation of modeled data
8 representational symbol
9 substitutability of equals for equals
10 free from convoluted interactions
11 philosophical reductionism
12 testable hypotheses
13 schematic description
14 representation in a medium
15 theoretical framework
16 linguistic expression or representation

6. Fill in the box on the left with the number of the most appropriate choice from the right.

(a)		separate compilation
(b)		abstract data type
(c)		coupling
(d)		opaque type
(e)		transparent
(f)		representation independence
(g)		information hiding
(h)		cohesion
(i)		specification
(j)		module
(k)		interface

1	a type with operations
2	incremental translation
3	limiting access
4	clearly visible
5	isolation of details
6	construct for encapsulation
7	visible representation
8	unity
9	boundary between systems
10	without specific instances
11	hidden representation
12	interaction
13	description

7. Why study programming languages?
8. Why does Eric Meijer like Visual Basic?
9. Define *orthogonality* in the context of programming language design.
10. What is the difference between a compiled and an interpreted language? Which one is Java?
11. What is a JIT compiler?
12. What the difference between what a JIT compiler does and dynamic linking?
13. If a language has garbage collection, must it necessarily be compiled or can it be interpreted?
14. What is the difference between an interactive language system and an interpreter?
15. What is an *API*?
16. What is a byte-code interpreter?
17. What is an abstract machine?
18. What language has been the most influential language? Justify your answer.
19. How was Ada developed?
20. What is the difference between syntax and semantics?
21. What is a formal language?
22. Give three different ways to describe a formal language.
23. What are the four levels of the Chomsky hierarchy?
24. Give a regular expression to describe strings with more than one pair of consecutive a's.
25. What languages over the alphabet $\{a, b, c, d\}$ do the following regular expression represent? Describe in words and give examples.
    (a) $(a + b)^*$

    (b) $((a \cdot b) + (c \cdot d))$

    (c) $((a + b)^* \cdot c)$

(d) $(((a \cdot b) + (c \cdot d))^* \cdot c)$

(e) $\emptyset^*$

26. Give some meta-characters used in extended regular expressions and describe precisely what each one means.

27. Give a specific example of a string and regex where it makes a difference whether or not a quantifier is greedy.

4th, section 2.1. Compare and contrast regular expressions and BNF as means to describe formal languages?

28. Give a BNF definition for expressions in some hypothetical programming language. Use a simple, but typical definition for expressions.

h, section 2.1.3. What is an ambiguous grammar?

29. Show that the following grammar with nonterminals S, A, and I is ambiguous:

$$\begin{array}{lcl} S & \rightarrow & A \\ A & \rightarrow & A \times A \mid I \\ I & \rightarrow & a \mid b \mid c \end{array}$$

30. (10 points.) Show that the following grammar with nonterminals S, and A is ambiguous:

$$\begin{array}{lcl} S & \rightarrow & A \\ S & \rightarrow & A\ A \\ A & \rightarrow & A\, a \\ A & \rightarrow & a \end{array}$$

31. (10 points.) Show that the following grammar with nonterminals S, and A is ambiguous:

$$\begin{array}{lcl} S & \rightarrow & A\ S \\ S & \rightarrow & a \\ A & \rightarrow & a \end{array}$$

32. Consider the following grammar with nonterminals S, A, and B:

$$\begin{array}{lcl} S & \rightarrow & A\, a\, B\, b \\ A & \rightarrow & A\, b \mid b \\ B & \rightarrow & a\, B \mid a \end{array}$$

Circle "valid" for each string that is in the language generated by this grammar. If it is not valid, circle "not valid."

(a)	baab	valid / not valid
(b)	bbbab	valid / not valid
(c)	bbaaaa	valid / not valid
(d)	bbaab	valid / not valid

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

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

35. Are regular expressions as expressive as CFG's?

36. Can the syntax of a typical programming language be expressed using regular expressions?

37. What are the tools *lex* and *flex*?

38. What is a scanner or lexical analyzer?

39. What are the tools *yacc* and *bison*?

40. What is a parser?

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

42. What is the relationship between a parse tree and a derivation?

43. What is a (concrete) parse tree? What is an abstract syntax tree?

44. What is the difference between a (concrete) parse tree and an abstract syntax tree?

45. Write a recursive descent parser for ...

46. What is the formal definition of a grammar?

47. What do you think of Python, Occam and Haskell's requirement to use proper indentation in statements? Would you expect this convention to make program construction and maintenance easier or harder? Why?

48. (12 points.) Fill in the boxes on the left with the most appropriate number from the right.

(a)	$\boxed{}$	denotational	(d)	$\boxed{}$	axiomatic
(b)	$\boxed{}$	operational			
(c)	$\boxed{}$	natural			

1 defined in terms of rules for evaluation

2 defined in terms of Post systems

3 defined in terms of rules relating states

4 defined in terms of attribute grammars

5 defined in terms of mathematical objects

6 defined in terms of “small-step” transition

7 defined in terms of an abstract machine

49. What are the three major kinds of formal semantics in the study of programming languages?

50. What is denotational semantics? What are its strengths and weaknesses as compared to other approaches to semantics.

51. What is axiomatic semantics? What are its strengths and weaknesses as compared to other approaches to semantics.

52. What is structural operational semantics?

53. What is natural semantics?

54. Operational semantics has been described as “mathematically weak.” What does this mean? Is this true?

55. What are the advantages and disadvantages of programming using assertions?

56. What is the difference between an *assertion* and a *condition*?

57. What is an *assertion*?

58. What is an *precondition*?

59. What is an *postcondition*?

60. (10 points.) Give a rule of inference for valid Hoare triples of the form shown below. Explain.

$$\{\,P\,\}\ S_1\textbf{;}\ \textbf{if}\ B\ \textbf{then}\ S_2\ \textbf{else}\ S_3\ \{\,Q\,\}$$

61. Explain what the preconditions and postconditions of a Hoare triple mean.

62. What is a *loop invariant*?

63. Compute the weakest precondition for each of the following assignments and postconditions (please simplify):

    (a) a=2*(b-1)-1$\{a > 0\}$

    (b) a=a-1$\{a > 0\}$

64. Consider the following loop:

```
while (a<z) {
  a := a+1;  b := b*c;
}
```

Part I. What is a loop invariant?

Part II. What properties does a loop invariant for the specific loop above have?

Part III. Which of the following are loop invariants for it:

(a) yes / no $true$

(b) yes / no $false$

(c) yes / no a

(d) yes / no $a > b$

(e) yes / no $a > 0$

(f) yes / no $z > 0$

(g) yes / no $a = a + 1$

(h) yes / no $b = c^a$

(i) yes / no $a = b^c$

Part IV. Which of the putative invariants in the previous part is the *best* invariant? Why?

65. What is aliasing? Give examples of constructs in two different languages that give rise to aliasing. Explain why aliasing is harmful.

66. Some languages have tried to eliminate or reduce aliasing. Why is it difficult to eliminate aliasing?

67. What are associativity and precedence?

68. Suppose you are designing a new programming language and are considering the syntax of identifiers. What are the arguments for and against making identifiers case-sensitive?

69. Most programming languages have only one syntax for all identifiers, for example, starts with a letter and continues with a letter or digit. Several programming languages have (as specified in the language definition) syntactically different kinds of identifiers. Name one such language, and give examples of the syntax, and the different kinds.

70. What is *binding time*?

71. Explain the distinction between decisions that are bound statically (at compile time) and those that are bound dynamically (at run time).

72. What is the advantage of binding things as early as possible? What is the advantage of delaying bindings?

73. What is the trade-off involved with early versus late binding?

74. What is a *keyword*?

75. What is a *reserved word*?

76. What is a *declaration*?

77. What is an *expression*?

78. What is a *statement*?

79. What is a *literal*?

80. What is a *constant*? What are the two types of constants?

81. What advantages are there to using constants?

82. Compare and contrast the three different kinds of "constants" illustrated below.

```
const pi = 3.14159;                             (* I. Pascal. manifest constant *)
S: constant String := Command_Line.Argument(1); -- II. Ada.    read only
final StringBuffer sb = new StringBuffer ();   // III. Java.   single assignment
```

83. What is a *dangling reference*?

84. A (compile-time) requirement that an identifier must be declared before it is used in the program cannot be enforced in typical programming languages. Give a common scenario in which there is no way this requirement can be met. What possible strategies can programming languages take?

85. Why are forward declarations necessary?

86. What is the difference between *scope* and *extent*?

87. Write a subprogram that determines the order of operand evaluation.

88. The assignment statement `A:=B` can mean different things. Explain carefully the different possible meanings it can have.

89. What are unary, binary, and ternary operators? Give an example of each from a specific programming language.

90. What are *aggregates*? Why are they useful?

91. In what specific ways do languages treat compound or structured data (like arrays and records) as units?

92. In APL all operators have the same precedence. What are the pros and cons of this approach?

93. Why don't issues of associativity and precedence arise in PostScript or Forth?

94. What does it mean for a language to be *referentially transparent*?

95. What is an *l-value*? An *r-value*?

96. What are the three storage classes?

97. Runtime storage management of FORTRAN (before FORTRAN 90) is quite simple. Explain.

98. What is an activation record?

99. What is the runtime organization in memory of a program in a typical ALGOL-like language?

100. What is the heap?

101. Name specific languages that do not permit the programmer to explicitly delete dynamically-allocated memory.

102. Why is the distinction between *mutable* and *immutable* values important in the implementation of a language with a reference model of variables?

103. What is an immutable object? What is the importance of immutable objects? In C++ or Java describe how to design classes for immutable objects.

104. C++ already has pointers from C, so what's the purpose of references?

105. What's the purpose of `const` references in C++?

106. If you have a `const` pointer in C++ can you change the object it points to? Explain.

107. What are the consequences of an "address of" operator like `&` in C/C++?

108. What are the important, common purposes of pointers?

109. Does Java have pointers?

110. What is the output of the following C program fragment:

```
int list [5] = {1, 3, 5, 7, 9};
printf ("%d\n", list[3]);
```

111. What is the output of the following Ada program block:

```
declare
   type Integer_Array is array (2..6) of Integer;
   List: Integer_Array := (1, 3, 5, 7, 9);
begin
   Integer_Text_IO.Put (List(3));
   Text_IO.New_Line;
end
```

112. Explain the notion of *definite assignment* in Java.

113. What are the advantages of updating a variable with an *assignment operator*, rather than with a regular assignment in which the variable appears on both the left- and right-hand sides?

114. Why do most languages leave unspecified the order in which the arguments of an operator or function are evaluated?

115. What is *short-circuit* Boolean evaluation? Why is it useful?

116. How does operand evaluation order interact with functional side effects?

117. Write a program that determines the order in which the operands (arguments to functions) are evaluated. Your program should print either "left then right" or "right then left."

118. You want to know the order in which the expression

     X + Y + Z

     is evaluated in some language. Write a simple program that prints the appropriate one of the following lines, or a similar line:

```
X first; Y second; Z third
Z first, Y second; X third
Y first, Z second; X third
```

119. Describe three common uses of the `goto` statement, and show how to avoid them using structured control-flow alternatives.

120. What is *garbage collection*?

121. Discuss the comparative advantages of *reference counts* and *mark-and-sweep collection* as a means of garbage collection.

122. Describe *generational garbage collection.*

123. Describe *conservative garbage collection.*

124. Do dangling references and garbage ever arise in the same programming language? Why or why not?

125. What is the purpose of both pointers and references in C++?

126. What is the loop/exit statement?

127. Describe the FORTRAN `DO` statement.

128. Describe the FORTRAN `FORALL` statement.

129. What is the definition of *control structure*?

130. What advantage does Ada's `exit` statement have over C's `break` statement?

131. What is an iterator? What languages have them?

132. All imperative languages have a `for` loop that executes a predetermined number of times. Some small differences in the `for` exist between different languages. Explain clearly what some of these differences are.

133. Despite the obvious drawback of C/C++ switch/case statement, Java, unlike C#, chose to include the switch statement exactly as it was in C/C++. Why?

134. What is a non-deterministic control construct? Give an example. Explain the importance.

135. At what point do these two summations differ from the mathematical ideal?

```
long i=0;
for (long j=0; ; j++) {
   i = i+1;
}

float f=0.0f;
for (long j=0; ; j++) {
   f = f+1.0;
}
```

136. What purpose do types serve in a programming language?

137. What is a type insecurity? Why is it bad? Give an example from Pascal (or elsewhere).

138. What is static type-checking? What is dynamic scoping? Is it possible to do static type checking with dynamic scoping? Why, or why not? Give an example.

139. What is the definition of a data type in a programming language? What does it mean for a language to be *strongly typed*? *Statically typed*? What prevents, say, C from being strongly typed?

140. What does *type equivalence* mean?

141. What differences are there between languages with regard to type equivalence?

142. What language feature, found in Modula-3 allows the program to attain name equivalence even though the language uses structural equivalence.

143. Can you write a program causing a type insecurity in the programming language C? in Ada?

144. What is polymorphism? What kinds of polymorphism are there?

145. What is an overloaded subprogram?

146. Java does not permit overloading of built-in operators. Is this a good decision or not? Explain.

147. Covariance is when a type operator respects the subtype relation. Contravariance is when a type operator reverses the subtype relation. Are arrays in Java covariant, contravariant, or neither? Illustrate using an example. Is this useful? Explain. Is this type-safe? Explain.

148. Define and explain the relationship between *default parameters* and *overloading*.

149. Define and explain the relationship between *overloading* and *dynamic dispatch*

150. Do you agree or disagree that position-independent arguments are pointless since all procedures ought to have just a few parameters anyway? Explain.

151. Is Perl a strongly-typed language? Explain.

152. What makes Perl a good language for doing report generation?

153. What is the difference between pass-by-reference and pass-by-value-result?

154. Write a simple program that distinguishes pass by reference and pass by value result. Explain.

155. Write a simple program that distinguishes call by reference and call by name. Explain.

156. Some languages permit assignment to call-by-value parameters in the body of the subprogram. Is this a good idea? Explain.

157. Consider two separate, independent executions of the following Ada-like program. Assuming that `X` is passed by value, what are the values of `I` and `A` after the call? Assuming that `X` is passed by reference, what are the values of `I` and `A` after the call?

```
PP: declare
    -- declare an array of 5 elements
    A: array (1..5) of Integer := (1,2,3,4,5);
    I: Integer := 1;
    procedure P (X: Integer) is
    begin
        X := 0;  I := 2;  X := 6;
    end P;
begin
    P (A[I]); -- call P
    -- value of "I", values of "A"?
end PP;
```

158. (10 points.) Consider the following program written in C syntax (arrays are 0 indexed):

```
int list [5] = {1, 3, 5, 7, 9};

void main () {
  int value=2;
  proc (list[value]);
}

void proc (int a) {
  list[2] = list[4];
}
```

For each of the following parameter-passing methods, what are all of the values of the array `list` after the call? Explain.

(a) pass by reference

(b) pass by value-result (also known as copy-in/copy-out)

159. Write a simple, Algol-like program that distinguishes dynamic scoping from static scoping. Explain.

160. Write a Python program (without global variables) to demonstrate whether Python uses static scoping or dynamic scoping.

161. What is an activation record?

162. What is a (potential) call graph?

163. Write a simple Algol-like program with non-local variable access.

164. Consider the following Ada-like program.

```
procedure Main is
   type F is access procedure (X: Integer); -- procedure type
   procedure R (X: Integer) is begin null; end R;
   procedure Q (FP: F) is begin FP(5); end Q;
   procedure P is
      N: Integer;
      procedure T (X: Integer) is begin N:=X; end T;
   begin
      Q (P'Access);                          -- pass procedure
      Q (T'Access);                          -- pass procedure
   end R;
begin
   P;
end Main;
```

Fill in the blank with the best answer.

(a) The static parent of procedure P is $\boxed{}$.

(b) The static parent of procedure Q is $\boxed{}$.

(c) The static parent of procedure R is $\boxed{}$.

(d) The static parent of procedure T is $\boxed{}$.

(e) The static depth of the only declaration of N is $\boxed{}$.

(f) The static depth of the only use of N is $\boxed{}$.

(g) The static distance of the only use of N is $\boxed{}$.

165. Consider the Ada-like program in the previous problem. Assume that the program compiles and executes as expected, show the runtime stack after every procedure call. Does the static link always equal the dynamic link during the execution of this program? Explain.

166. Two steps are needed in locating any non-local variable in a statically scoped language implemented using activation records. What are these two steps?

167. Write a simple Algol-like program in which the static chain does not equal the dynamic chain. Trace the execution of the program.

168. What are the advantages and disadvantages of using a stack of activation records to provide for variable access in block-structured programming languages?

169. What is the purpose of the display in the implementation of ALGOL-like languages?

170. Explain how passing procedures as arguments is implemented in a block-structured language?

171. What is a procedure closure?

172. What is the difference between a procedure closure and an activation record?

173. How does C implement functions that return functions?

174. Why do some programming languages forbid returning subprocedures? Yet the language C does not, explain.

175. If you were designing a new programming language would you allow assignment of subprocedures to variables? Why or why not?

176. Suppose you are writing a compiler for a language like Modula-3 which has procedure variables. In the program `Main`, `x` is a procedure variable.

```
MODULE Main;
    TYPE f = PROCEDURE (z: INTEGER); (* procedure type *)
    VAR x: f;
    PROCEDURE P (z: INTEGER) = BEGIN (* ... *) END P;
BEGIN
    x := P;    (*  assignment  *)
    x(2);      (*  call        *)
END Main.
```

Explain exactly how the assignment gets implemented and how the call gets implemented so that non-local variable access works correctly.

177. Suppose you are writing a compiler for a language like Modula-3 which has procedure variables. In the program `Main`, `x` is a procedure variable.

```
typedef void (*f)(int);  // procedure type
void p (int n) {   /*  */ }

int main (int argc, char *argv[]) {
   f x = &p;  // assign procedure p to variable x
   x (2);     // call whatever is in x
   return 0;
}
```

Explain exactly how the assignment gets implemented and how the call gets implemented so that non-local variable access works correctly.

178. How has the implementation of non-local variable access influenced programming language design?

179. Illustrate exactly what Michael Scott means in the following sentence:

First-class subroutines in a language with nested scopes introduce an additional level of complexity: they raise the possibility that a reference to a subroutine may outlive the execution of the scope in which that routine was declared.

180. What is an *exception handler*?

181. Since exception handling is not necessary in a language, explain why most modern languages have it.

182. Name some differences in the way exception handling is done in different languages. Give examples from specific programming languages (do not use PL/I).

183. What are the differences between the C++ **throw** specification and the Java **throws** specification?

184. In every language since PL/I, exception propagation is essentially the same. Describe exception propagation as in, for example, ML, Ada, C++, Modula-3, and Java.

185. Exception handling in ML, Ada, C++, Modula-3, and Java are pretty much alike. Describe some differences.

186. Consider the following Java program in which an exception may be raised in procedure `proc_A` at the point indicated in the program.

```
class X {

   static void proc_A() {
      try {
         System.out.println ("begin proc_A");
         /*
            throw NullPointerException, RuntimeException, Exception, et
         */
      } catch (NullPointerException e) {
         System.out.println("handler in proc_A");
      } finally {
         System.out.println ("end proc_A");
      }
   }

   static void proc_B() {
      try {
         System.out.println("begin proc_B");
         proc_A();
      } catch (RuntimeException e) {
         System.out.println("handler in proc_B");
      } finally {
         System.out.println("end proc_B");
      }
   }

   public static void main (String args[]) {
        try {
            System.out.println("begin main");
            proc_B();
        } catch (Exception e) {
            System.out.println("handler in main");
        } finally {
            System.out.println("end main");
        }
    }
}
```

Describe the execution of the entire program for each of the following scenarios by showing the output and saying how the program terminates.

(a) Suppose procedure `proc_A` raises no exception.

(b) Procedure `proc_A` raises the exception `RuntimeException`.

(c) Procedure `proc_A` raises the exception `NullPointerException`.

(d) Procedure `proc_A` raises the exception `Exception`.

187. Consider the `finally` clause in either Modula-3 or Java. What are all the different circumstances in which the `finally` clause is executed?

188. What is a variant record? What good are they? Give a scenario in which a variant record is appropriate. How are variant records allocated at runtime?

189. What is information hiding?

190. Pick some language in which the representation of a data type can be hidden. Name the language and give an example.

191. All data types exported from a module in Modula-2 must be a pointer type. Why was this restriction made?

192. Concerning data types, what is representation independence and is it good or bad?

193. What are opaque and transparent data types?

194. What is meant by a module?

195. What is meant by encapsulation?

196. What is meant by aggregation?

197. What is meant by representation independence?

198. Define *separate compilation*. Explain.

199. What is the advantage of separate compilation?

200. What is a specification?

201. Write a very short C or C++ program that demonstrates that these languages do no inter-module type checking.

202. Does C or C++ check the data types of separately compiled objects?

203. What is meant by the client of a module?

204. What is the difference between an abstract data type and a module?

205. Object-oriented languages often have pseudo-variables (to use Smalltalk terminology) called `this` and `super`. What is the purpose of each?

206. What is an object-oriented programming language?

207. What is a *class* in an object-oriented language? What is an *instance* of an object?

208. What objected-oriented features are different in Java than in C++?

209. Describe the support for object-oriented programming in Ada 95.

210. Often subtype polymorphism is used in place of parametric polymorphism in object-oriented languages. Describe a scenario in which this might occur and explain. Are there any drawbacks to this approach?

211. What is dynamic dispatch in object-oriented languages? How is it implemented? In designing a program, when would you use dynamic dispatch?

212. What is the purpose of the keyword `virtual` in C++?

213. Why is use of `instanceof` (to determine the class of an instance of an object) in Java an indication of poor object-oriented design in some cases?

214. Some object-oriented languages have multiple inheritance. Yet the designers of Java chose single inheritance. Please explain the issue and explain the rational of the Java designers.

215. What is meant by *narrowing* and *widening*? What is the difference?

216. What is referential transparency?

217. The primitive function `QUOTE` plays an important role in LISP and Scheme. Does a similar function exist in SML or Haskell? Please circle the answer: yes / no. Explain.

218. Arrays play an important role in imperative languages. Do array play an important role in functional languages? Please circle the answer: yes / no. Explain.

219. LISP uses a simple, uniform syntax called Cambridge Polish. Describe this syntax.

220. What does the following Scheme function do?

```
(define (y s list)
   (cond
      ((null? list) () )
      ((equal? s (car list)) list)
      (else (y s (cdr list)))))
```

221. What does the following lambda expression compute: $\lambda(x)x \times x \times x$

222. The “i” in GHCi stands for

    (a) interesting
    (b) interpreter
    (c) interactive
    (d) internal
    (e) interjection

223. In Haskell what is the usual name given to the symbol “\”?

    (a) backslash
    (b) slosh
    (c) back-slat
    (d) whack
    (e) lambda
    (f) none of the above

224. What is the usual name given to the Haskell operator denoted by “:”?

    (a) colon
    (b) cons
    (c) double point
    (d) two-spot
    (e) cdr (“could-der”)
    (f) none of the above

225. What is the algorithmic complexity of the Haskell `++` operator?

    (a) $O(1)$
    (b) $O(n)$ where n is the length of the first argument
    (c) $O(m)$ where m is the length of the second argument
    (d) $O(n^2)$ where n is the length of the first argument
    (e) $O(n+m)$ where n is the length of the first argument and m is the length of the second argument

226. What is a higher-order function? Give an example.

227. What does the following ML function do?

```
datatype T = n | lf of int * T * T;
fun f n = [] | f (lf (x,l,r)) =  (f l) @ (x :: (f r));
```

228. Rewrite the following expressions (which are syntactically correct in SML and Haskell) to equivalent ones in SML or Haskell without using square brackets `[]`.

(a) `[]`	
(b) `[[]]`	
(c) `[[[]]]`	
(d) `[()]`	
(e) `([])`	
(f) `[1,2,3]`	
(g) `["a","b"]`	
(h) `[(1,2,3)]`	
(i) `[[],["a",""]]`	

229. What are the SML types of each of the following:

```
val a = [1,2,3];
val b = [];
val c = (fn x=>x+3);
val d = (4.5, "a");
val e = (fn (x,_) => (x,x,x));
```

230. Given

```
real : int -> real
Math.cos : real -> real
Int.min : int * int -> int
```

What are the SML types of each of the following expressions:

```
fun a (f,g) x = g (f x);
val b = a (real, Math.cos);
val c = a (Int.min, fn x=>x+1);
```

231. Write the factorial function in SML.

232. Consider the following SML function definition.

```
fun map f nil = nil |
    map f (h::t) = f(h) :: (map f t);
```

(a) What is the type of the function?

(b) Describe in a few words what the function does.

233. Which of the following SML types indicates the best design for the map function? Explain.

```
('a->'b) * 'a list -> 'b list
'a list * ('a->'b) -> 'b list
('a->'b) -> 'a list -> 'b list
'a list -> ('a->'b) -> 'b list
```

234. What is the type of the following ML function? Describe in a few words what the function does.

```
fun filter P nil    = nil |
    filter P (h::t) = if P(h) then h::(filter P t) else filter P t;
```

235. Is the `datatype` binding in SML opaque or transparent? Explain.

236. What is the type of the ML function `f` below? Describe in a few words what the function does.

```
datatype T = nl | nd of int * T * T;
fun f nl = [] | f (nd (x,l,r)) =  (f l) @ (x :: (f r));
```

237. What is type inference?

238. What can we say about the SML or Haskell expression e of the form $exp_1 exp_2 exp_3$, if it is syntactically and semantically correct.

(a) true / false e is a list of some type.

(b) true / false exp_1 must be an identifier.

(c) true / false exp_2 cannot be an identifier.

(d) true / false exp_3 cannot be a function.

(e) true / false exp_1 cannot be a list.

(f) true / false exp_2 cannot be a list.

(g) true / false exp_2 cannot be of the form `a->b`.

(h) true / false the type of exp_1 is of the form `a->b`.

(i) true / false the type of exp_1 is of the form `a->b->c`.

(j) true / false e is exactly equivalent to $(exp_1 exp_2) exp_3$.

(k) true / false e is exactly equivalent to $(exp_1(exp_2)exp_3)$.

(l) true / false e is exactly equivalent to $exp_1(exp_2 exp_3)$.

239. What is the type of the following ML function? How does ML infer the type? Describe in a few words what the function does.

```
fun f x nil    = false |
    f x (h::t) = x=h orelse (f x t);
```

240. Consider the following Haskell data type and function, and answer the questions below:

```
data T a = LF a | N (T a) (T a) (T a)
f g (LF x)     = x
f g (N l c r) = g (f g l) (g (f g c) (f g r))
```

(a) What does it mean for a data type to be polymorphic?

(b) Is the data type `T` polymorphic?

(c) What is the use of the identifier `a` in the Haskell code?

(d) If possible, give an example of a value that has type `T Int`.

(e) If possible, give an example of a value that has type `T Char`.

(f) Give some other (illuminating) example of a value involving type `T`.

(g) What is the name of the function defined by the programmer in the Haskell code?

(h) Can a Haskell function have more than one type? yes / no. Explain, give an example. If a function can have more than one type, is there a best type? yes / no. Explain, give an example.

(i) What is the type of the following function?

(j) Apply the function to some value. What is the result?

(k) Describe what the function does.

241. The two kinds of functional languages are eager and $\boxed{}$. Give the name of a language of each kind.

242. Must all ML functions be defined using the constructs `fun` or `fn`? Please circle the answer: yes / no. Explain using an example.

243. What is the difference between the SML `let` construct and the `local` construct?

244. What is the difference between an expression and a declaration in SML? Give examples of each.

245. What is a higher-order function? In any functional language, give an example.

246. Does Haskell or ML do garbage collection? Does a Haskell or ML implementation require a heap? Explain.

247. Does the nature of the Haskell programming language require garbage collection (by the runtime system)? Why, or why not? Does Java? Does C++? Does Prolog?

248. What is a functor?

249. Explain the significance of functors in PROLOG.

250. What construct in Haskell (or ML) corresponds to a functor in PROLOG? Explain.

251. Is there any construct in Haskell (or ML) that resembles a functor in PROLOG? Explain.

252. What construct in PROLOG corresponds to a constructor in Haskell (or ML)? Explain.

253. Why can't all functions be used in patterns in Haskell (or ML)?

254. What role do PROLOG functors play in unification?

255. What role do PROLOG relation symbols play in unification?

256. Why does Haskell not have exception handling, but ML does?

257. Name an imperative programming language: $\boxed{}$. Name a functional programming language: $\boxed{}$. Name a logic programming language: $\boxed{}$.

258. What does it mean for a language to be *nonprocedural*?

259. What are the two or three different kinds of input to PROLOG?

260. What are the two or three different kinds of output does PROLOG give in response to a query?

261. Compare and contrast pattern matching in ML with pattern matching in PROLOG?

262. Are there functions in PROLOG? How do you define a function $f(x) = y$ in PROLOG?

263. What is the result of the following PROLOG program:

```
father(bob).
man(X) :- father(X).
```

given the query `man(X)`.

264. Define *unification*.

265. Define *resolution*. What does it have to do with PROLOG?

266. What is meant by the *closed world assumption* in PROLOG?

267. Formulate in PROLOG the classical syllogism:

    All men are mortal;
    Socrates is a man;
    Therefore Socrates is mortal.

268. Formulate in PROLOG the syllogism:

> All students hate exams;
> This is an exam;
> Therefore I hate this exam.

269. Given the relations

`Father(x,y)`	`x` is the father of `y`
`Mother(x,y)`	`x` is the mother of `y`
`Female(x)`	`x` is female
`Male(x)`	`x` is male

define (pure) PROLOG relations for the following notions:

(a) `x` is a sister of `y` (same parents)

(b) `x` and `y` are siblings (brother or sister)

(c) `x` is a grandson of `y`

(d) `x` is a descendant of `y`

270. What is a unifying substitution? Give the most general unifying substitution for each of the following pairs of terms (x, y, and z are variables):

$$\begin{array}{rl} g(x,a) & g(x,a) \\ g(x,y) & g(y,h(a,x)) \\ f(g(a,b),h(x,y)) & f(g(z,b),h(b,b)) \end{array}$$

271. Consider the following PROLOG clauses (`x`, `h`, `t`, and `z` are variables):

```
A([],x,x).
A([h|t],x,[h|z]) :- A(t,x,z).
```

What are the answers to the following queries (`x`, and `y` are variables)? If there is more than one solution, give any two of them.

(a) `A([E],[],[E])?`

(b) `A([E],[],[E,F])?`

(c) `A([],[],x)?`

(d) `A([E],[],x)?`

(e) `A([E],[F],x)?`

(f) `A([E,F,G],[H,I],x)?`

(g) `A(y,[H,I],[G,H,I])?`

(h) `A(x,y,[E,G,I])?`

(i) `A([x|y],[H,I],[G,H,I])?`

(j) `A([x|F],[y|H,I],[E,F,G,H,I])?`

272. Define "append" of two lists in PROLOG. Define the member function in PROLOG that tests if an element is a member of a list.

273. Give a very simple Prolog program and a single query which has an infinite number of solutions. Show the search space.

274. Consider the following PROLOG database of nullary predicates:

```
A :- B,A.
A :- B.
B.
```

Show the entire search space for the query `A,B?`.

275. Consider the following PROLOG database of nullary predicates:

```
A.
A :- B.
A :- C.
B.
C.
```

Show the entire search space for the query `A,B,C?`. Is the search space finite? Please circle the answer: yes / no. How many solutions are there? $\boxed{}$

276. Consider the following PROLOG program where `A`, `B`, `C`, and `D` are unary predicate symbols; `a`, `b`, `c`, and `d` are nullary functor symbols; and `x`, `y`, and `z` are variables:

```
A(y) :- C(y),D(y).
B(x) :- A(d).
B(x) :- C(c).
C(a) :- B(b).
C(d).
C(z) :- B(z).
D(d).
```

Show the entire search space for the query `B(x),C(a)?`. Is the search space finite? Please circle the answer: yes / no. How many solutions are there? $\boxed{}$

277. Consider the following PROLOG database:

```
Ancestor(x,y) :- Parent(x,z), Ancestor(z,y).
Ancestor(x,y) :- Parent(x,y).
Parent(A,C).
Parent(B,C).
```

Show the entire search space for the query `Ancestor(A,C)?`. How many solutions are there?

278. Suppose you are given a collection of "parent" facts. For example,

```
Parent (George, Edward).  // A parent of George is Edward
Parent (Edward, Victoria).// A parent of Edward is Victoria
Parent (Edward, Albert).  // A parent of Edward is Albert
```

Consider the following two PROLOG programs (`x`, `y`, and `z` are variables). The only difference is that the clauses in the tail of the second rule are reversed.

```
Ancestor(x,y) :- Parent(x,y).
Ancestor(x,y) :- Parent(x,z), Ancestor(z,y).


Ancestor(x,y) :- Parent(x,y).
Ancestor(x,y) :- Ancestor(z,y), Parent(x,z)
```

What difference does the reversal make in respect to the search space of a query? In particular, are there any differences in the solutions? Which is a better PROLOG program and why?

279. Consider the following Haskell function:

```
f [] = []
f [_] = []
f (x:_:rest) = x : (f rest)
```

(a) What is the type of the Haskell function?

(b) Describe in plain words what the function does.

(c) Translate the function into PROLOG.

280. The list data structure is primitive in PROLOG. Nonetheless, explain how the programmer could define the list data structure in PROLOG anyway.

281. What is the occurs check in unification. What role does the occurs check play in PROLOG? Explain.

282. Give two reasons why PROLOG cannot be used as a theorem prover.

283. Scott states on page 613 in the 4th edition of his textbook:

> In the abstract, logic programming is a very compelling idea: it suggests a model of computing in which we simply list the logical properties of an unknown value, and then the computer figures out how to find it. Unfortunately, reality falls quite a bit short of the vision.

Please explain:

(a) What is the compelling vision? Why is it so attractive?

(b) What are the theoretical and practical reasons the vision is unattainable?

284. The Beach Boys were formed in 1961 by brothers Carl, Dennis and Brian Wilson and their cousin Mike Love. Throughout the 1960s the Beach Boys sang about surfing, girls and hot rods. Their unique style of vocal harmonization profoundly altered the direction of rock and roll. The Beach Boys epitomized the all-American image with their well-groomed, suntanned surfer look, creating the "California Myth" that ultimately lured droves of teenagers to the west coast looking for endless summer days and "two girls for every boy."

"Surf City"
Jan Berry and Brian Wilson

And we're goin' to Surf City, 'cause it's two to one
You know we're goin' to Surf City, gonna have some fun
You know we're goin' to Surf City, 'cause it's two to one
You know we're goin' to Surf City, gonna have some fun, now
Two girls for every boy

Assert in PROLOG that there are two girls for every boy.