Section A: Programming Languages Miscellany

1. The first widely-used functional programming language was b.
   a. FORTRAN b. LISP c. SNOBOL d. COBOL e. C

2. Programs written in c tend to be very concise (i.e., short) but also difficult to read.
   a. FORTRAN b. LISP c. APL d. Pascal e. Ada

3. a was designed specifically to serve as the programming language of the U.S. Department of Defense.

4. a, which is still widely used for developing software for scientific and engineering applications, was the first widely used high level programming language.
   a. FORTRAN b. COBOL c. LISP d. C e. Algol-60

5. Having been the leader of the team who developed FORTRAN, d twenty years later gave an ACM Turing Award speech in which he encouraged language designers to break the “von Neumann bottleneck”.

6. The class construct (to support data abstraction) first appeared in e.
   a. FORTRAN b. COBOL c. Pascal d. C++ e. Simula 67

7. The language that may achieve the highest degree of orthogonality (for good or ill) is c.

8. b, the designer of Pascal, is well known for advocating simplicity in programming language design.

9. e, much of whose syntax was borrowed by Java, is widely used in systems programming.
   a. Ada b. Smalltalk c. PL/I d. COBOL e. C

10. The a paradigm of programming languages is most closely associated to artificial intelligence applications.
    a. functional b. imperative c. object-oriented d. scripting

11. The language most closely associated to predicate calculus is c.

12. The b paradigm of programming languages provides a not-too-distant abstraction of the von Neumann architecture.
    a. functional b. imperative c. object-oriented d. logic

13. The data structure most closely associated to LISP is the a.
    a. list b. array c. associative array d. hash table
14. The language that first gave full support to object-oriented programming was e.
   a. Ada    b. Objective-C    c. Smalltalk    d. LISP    e. SNOBOL

15. a has powerful array-processing operators denoted by symbols that are not in the ASCII alphabet.
   a. APL    b. Ada    c. LISP    d. SNOBOL    e. PL/I

16. c was the first language whose syntax was described by the formal notation of a context-free grammar; among its descendants are Pascal and Ada.
   a. FORTRAN    b. LISP    c. Algol-60    d. COBOL    e. C

17. a is an HTML-embedded scripting language designed for doing server-side processing in Web applications.
   a. PHP    b. SNOBOL    c. C#    d. Python    e. JavaScript

18. e is an HTML-embedded scripting language widely used for doing client-side processing in Web applications.
   a. PHP    b. SNOBOL    c. C#    d. Visual Basic    e. JavaScript

19. b is one of Microsoft’s .NET languages.

20. b was designed in the early 1960’s for doing text processing and, in particular, pattern matching.
   a. Algol-60    b. SNOBOL    c. APL    d. PL/I    e. Simula 67

21. d, a rather complex language, was designed in the early 1960’s with the intent that it would be good for developing both scientific and business applications.
   a. Algol-W    b. SNOBOL    c. APL    d. PL/I    e. Simula 67

22. Designed for business applications, b has been estimated to be the programming language in which the greatest number of lines of code has been written.
   a. Java    b. COBOL    c. Pascal    d. Ada    e. FLOW-MATIC

23. d refers to the ability (in a programming language) to meaningfully combine various features in all possible combinations.

24. Variables, assignment, and iteration are attributes that are most indicative of the c paradigm of programming languages.
   a. functional    b. logic    c. imperative    d. object-oriented

25. d was the German engineer who not only built several computers during World War II but also developed the noteworthy programming language PlanKalkül, which, unfortunately, was unknown until 1972.
Section B: Language Evaluation Criteria and Other Concepts

Answer at least one of the questions in this section, and a total of at least five from this section together with Section C.

1. Programming languages can be implemented via compilation at one extreme or pure interpretation at the other, or via a hybrid approach somewhere in the middle. Identify and discuss the relevant tradeoffs between these approaches, and comment upon your relevant personal experiences, if you’ve had any.

**Answer:** Omitted for now.

2. Some programming languages (e.g., Java) have large standard libraries. Discuss how the assessment of a programming language (in terms of Sebesta’s criteria of readability, writability, and reliability) may be affected by the language’s standard library, or lack thereof.

**Answer:** Omitted for now.

3. Explain what role(s) lexical and syntax analysis play in the context of programming language translation/implementation.

**Answer:** Omitted for now.
Section C: Language Description and Recognition Mechanisms

Answer at least three of the five questions in this section, and a total of at least five from this section together with Section B.

1. Develop a context-free grammar that generates boolean expressions having as atomic operands the symbols \( t \) and \( f \) (for true and false, respectively) and binary infix operators \( \land \) and \( \lor \) (for conjunction/and and disjunction/or, respectively). Conjunction/and has higher precedence than disjunction/or. Both of these operators associate to the left. For a little extra credit, include the unary prefix \( \neg \) operator (for negation/not), which has higher precedence than the other operators.

Example: \( \neg t \land \neg (f \lor t \land t) \lor f \)

**Hint:** Don’t forget about our favorite grammar for arithmetic expressions. (It appears in Problem 2 of HW#2.)

**Answer:**

\[
\begin{align*}
E & \rightarrow E \lor T \\
E & \rightarrow T \\
T & \rightarrow T \land F \\
T & \rightarrow F \\
F & \rightarrow (E) \\
F & \rightarrow \neg F \\
F & \rightarrow t \\
F & \rightarrow f
\end{align*}
\]

2. The given context-free grammar generates some language over the alphabet \( \{a,b\} \). (It is not important to know exactly what language that is.) Augment the grammar to obtain a translation grammar (as in Homeworks #3 and #5) having the property that a string with \( k \) occurrences of \( a \) and \( m \) occurrences of \( b \) translates into \( 0^k1^m \) (i.e., the string comprised of \( k \) occurrences of 0 followed by \( m \) occurrences of 1). For example, the string bbabababb (which, indeed, is generated by this grammar and which has three occurrences of \( a \) and six occurrences of \( b \)) translates into 000111111 (i.e., \( 0^31^6 \)).

\[
\begin{align*}
S & \rightarrow AaSb \\
S & \rightarrow b \\
A & \rightarrow bAaS \\
A & \rightarrow bab
\end{align*}
\]

**Answer:** There is none! Sorry about that.

3. Consider the following attributed grammar. Show a fully-attributed parse tree for the string \( 5 + 7 \ast 3 \). That is, next to each non-leaf node in the parse tree show the value(s) of
its attribute(s). In this grammar, each nonterminal has one synthesized attribute and \( B \) has, in addition, one inherited attribute.

Also give, in English, a characterization of the language generated by this grammar and an indication as to what information is provided by the attributes (in particular, \( S.syn \)).

(1) \( S \rightarrow AB \) \( B.inh := A.val; \ S.syn := B.syn; \)
(2) \( B \rightarrow +AB \) \( B_1.inh := B.inh + A.syn; \ B.syn := B_1.syn; \)
(3) \( B \rightarrow *AB \) \( B_1.inh := B.inh \times A.syn; \ B.syn := B_1.syn; \)
(4) \( B \rightarrow \epsilon \) \( B.syn := B.inh; \)
(5) \( A \rightarrow 0 \) \( A.syn := 0; \)
(6) \( A \rightarrow 1 \) \( A.syn := 1; \)
\( \ldots \)
(14) \( A \rightarrow 9 \) \( A.syn := 9; \)

**Answer:** Here is the fully-attributed parse tree. (The label of the rightmost leaf, \( e \), denotes the empty string.) To the right of each interior node's label are the values of its attribute(s), in parentheses. In the case of \( B \)-nodes, the inherited attribute's value is listed first.

```
S(36)
 /\ 
 / \ 
 / \ 
 / \ 
A(5) B(5,36)
 | / \ 
 | / | \ 
 | / | | \ 
 | / | | | \ 
 | / | | | | 
| | | | | |
5 + A(7) B(12,36)
 | / \ 
 | / | \ 
 | / | | \ 
 | / | | | \ 
| | | | | |
7 * A(3) B(36,36)
 | | |
3 e
```

The language generated by the grammar consists of all arithmetic expressions in which the only operators are (binary) + and \( * \), the only atomic operands are one-digit numerals, and there are no parentheses anywhere. For any parse tree, the value of the (synthesized) attribute of the root is the value of the expression yielded by the tree, under the interpretation that neither operator has precedence over the other and that association is to the left. (That explains why \( 5 + 7 \times 3 \) is interpreted as \( ((5 + 7) \times 3) \), which gives 36.)
4. Develop a finite state machine that recognizes the set of strings over the alphabet \{a, b, c\} containing precisely those strings satisfying these three conditions:
(a) \(c\) occurs at least once
(b) Before the first occurrence of \(c\), every occurrence of \(a\) is followed immediately by an occurrence of \(b\), and
(c) After the first occurrence of \(c\), no occurrence of \(a\) is followed immediately by an occurrence of \(b\).

**Answer:** Below is an FSM, given in tabular form. It is left to the reader to produce the corresponding state graph.

\[
\begin{array}{ccc|c|c|c|}
\text{a} & \text{b} & \text{c} & \text{q0} & \text{q1} & \text{q2} & \text{q3} & \text{q4} \\
\hline
+ & + & + & q0 & q1 & q0 & q2 & \text{initial state} \\
+ & + & + & q1 & q4 & q0 & q4 & \text{q1} \\
+ & + & + & q2 & q3 & q2 & q2 & \text{accepting state} \\
+ & + & + & q3 & q3 & q4 & q2 & \text{accepting state} \\
+ & + & + & q4 & q4 & q4 & q4 & \text{q4} \\
\end{array}
\]

5. For the given context-free grammar, construct its parse table (as in HW#5) and indicate whether or not it satisfies the definition of a \(q\)-grammar.

1. \(S \rightarrow \epsilon\)
2. \(S \rightarrow bAS\)
3. \(S \rightarrow aBSc\)
4. \(A \rightarrow aA\)
5. \(A \rightarrow c\)
6. \(B \rightarrow \epsilon\)
7. \(B \rightarrow c\)
8. \(B \rightarrow dA\)

**Answer:** Here is the grammar’s parse table:

\[
\begin{array}{cccccc}
\text{a} & \text{b} & \text{c} & \text{d} & \$ & \\
\hline
S & (3) & (2) & (1) & - & (1) \\
A & (4) & - & (5) & - & - \\
B & (6) & (6) & (6,7) & (8) & - \\
\end{array}
\]
Due to the \((B, c)\) entry containing two different productions, the grammar is **not** a \(q\)-grammar. The most common error was to omit production (6) from the \((B, c)\) table entry. According to the rules for producing a parse table, for that production to be in that entry of the table requires that \(c\) be a member of \(\textit{FOLLOW}(B)\). Is that really the case? The answer is yes, due to productions (3) and (1). Specifically, we have the derivation

\[
S \overset{(3)}{\Rightarrow} aBSc \overset{(1)}{\Rightarrow} aBc$

that yields a sentential form in which \(c\) follows \(B\).