Assignment # 8 with answers

1) Languages. Matching. Use each language once.

(a) What programming language has dominated scientific computing over the past 40 years? **Fortran**

(b) What programming language has dominated business applications over the past 40 years? **COBOL**

(c) What programming language has dominated artificial intelligence programming over the past 40 years? **LISP**

(d) What language FIRST used orthogonality as a major design criterion? **Algol 68**

(e) In what language is UNIX written? **First B, then C**

(f) What language was the first to introduce some object-oriented concepts. **Simula**

(g) What is a language that performs type inference? **Statically typed languages/ML**

(h) What language has the slogan "Write once, run anywhere"? **JAVA**

(i) Which language is considered to be the first fully object-oriented language? **Smalltalk**

(j) What language did the US Department of Defense try to mandate for use in all of its software? **ADA**

ADA, ALGOL68, C, COBOL, FORTRAN, LISP, JAVA, ML, SIMULA, SMALLTALK

2) Paradigm. Matching.

(a) What is the name of the category of programming languages whose structure is dictated by the von Neumann computer architecture? **IMPERATIVE**

(b) A paradigm that allows specification of what has to be computed rather than just how a computation is to be carried out. **NON-PROCEDURAL**

(c) A paradigm incorporating encapsulation, inheritance, and dynamic type binding. **OBJECT-ORIENTED**

OBJECT-ORIENTED, IMPERATIVE, NON-PROCEDURAL

3) ___T___ Dynamic type checking adds some execution-time overhead, but improves reliability of programs.

4) ___T___ Attribute grammars can capture some features in a language that BNF grammars can not.
Scope and Lifetime (5)

5) What's the difference between scope and lifetime with regard to variable declarations?
A scope refers to the range of the program in which the variable can be referenced. Lifetime refers to
the period of time during which it can be referenced.

This is a Pascal (PL/SQL, ADA) like language.

```pascal
program test;
var a, b: integer;

procedure sub1;
  var a, c, e: integer;
begin
  a := 6; c := 7; e := 8;
  ... {position 3}
end; {sub1}

procedure sub2;
  var a, c, d: integer;
begin
  a := 3; c := 4; d := 5;
  ... {position 2}
  sub1();
  ...
end; {sub2}

begin
  a := 1; b := 2;
  ... {position 1}
  sub2();
  ...
end. {test}
```

(a) Determine what variables are visible at positions 1, 2, 3 and what their values are (given the call sequence indicated) with static scoping. (6)
(b) Determine what variables are visible at positions 1, 2, 3 and what their values are (given the call sequence indicated) with dynamic scoping. (6)

<table>
<thead>
<tr>
<th>Position</th>
<th>Static Scoping</th>
<th>Dynamic Scoping</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a=1, b=2</td>
<td>a=1, b=2</td>
</tr>
<tr>
<td>2</td>
<td>a=3, b=2, c=4, d=5</td>
<td>a=3, b=2, c=4, d=5</td>
</tr>
<tr>
<td>3</td>
<td>a=6, b=2, c=7, e=8</td>
<td>a=6, b=2, c=7, d=5, e=8</td>
</tr>
</tbody>
</table>
Specifying a Language.

This problem asks you to define a simple language using a deterministic finite automaton (DFA), a regular expression (RE) and a context-free grammar (CFG). The language is that consisting of a single email address. We won't cover the full syntax of valid email addresses specified by IETF RFC 822, but just simple ones like foo@bar.com and 331@cs.umbc.edu. Here's an English description of the kinds of email addresses we want to recognize: An email address is a USERNAME followed by the "@" character followed by a HOST. A USERNAME is one or more alphanumeric characters. A HOST is composed of two or more DOMAINs separated by the character ".". A DOMAIN is a sequence of one or more alphanumeric characters. To make it easier, assume that ALPHA represents any alphabetic character and DIGIT represents any digit.

(a) Draw a DFA for this language. Mark the accepting states with a double circle. Label each transition arrow with either an input character or one of the special tokens ALPHA or DIGIT.

(b) Write a RE to recognize this language using Lex's notation, again assuming that ALPHA represents any alphabetic character (e.g., [a-zA-Z] and DIGIT represents any digit (e.g., [0-9]).

(c) Recast your language as a CFG using the non-terminal EMAIL as the start symbol and whatever other non-terminals you want. Again, treat DIGIT and ALPHA as tokens, so you need not expand them.

\[
\text{DFA}
\]

\[
\text{RE: (alpha | digit)+ @ (alpha | digit)+ . ((alpha | digit)+)+}
\]

\[
\text{CFG:}
\]

Email -> Username @ Host
Username -> Alphanums
Host -> Alphanums . Alphanums | Alphanums . Host
Alphanums -> Alphanumeric | Alphanumeric Alphanums
Alphanumeric -> alpha | digit