0	00/
1	20/
2	05/
3	15/
4	15/
5	15/
6	20/
7	30/
8	30/
	150/

CMSC 331 First Midterm Exam

Name: _____Sample Answers_____

Student ID#:_____

You will have seventy-five (75) minutes to complete this closed book exam. Use the backs of these pages if you need more room for your answers. Describe any assumptions you make in solving a problem. We reserve the right to assign partial credit, and to deduct points for answers that are needlessly wordy.

0. Warm up (0 pts)

How many times in a 24 hour day do the minute and hour hands of a clock exactly overlap? Assume that the day starts at the stroke of midnight and goes up to, but doesn't include, the stroke of midnight 24 hours later.

22 times. The hands start out overlapped. It talks a bit more than 65 minutes before they overlap again. After 12 hours, they overlap again. So, in twelve hours they can't line up 12 times (the time is more than an hour), so it must be eleven times. Thus, the time between one overlap and another is 12/11 hours or 65 minutes and 45 seconds. So, in a 24 hour duration, there are (24 / (12/11))overlaps, or 22.

1. True/False (20 pts)

For each of the following questions, circle T (true) or F (false).

T **F** Large languages are more orthogonal than small languages.
T F FORTRAN introduced many control structures, but few data structures.
T **F** Reserved words are special words that are allowed to be redefined by the programmer.
T F In general, Perl programs are compiled and interpreted every time they are executed.
T **F** In general, Java programs are compiled and interpreted every time they are executed.
T **F** LISP and ML are often cited as examples of imperative programming languages.
T **F** Every attribute grammar can be rewritten as a BNF grammar with possibly more non-terminals and more rules.
T **F** A grammar G is ambiguous if there is more than one derivation for at least one sentence in the language defined by G.
T F A grammar G is ambiguous if there is more than one parse tree for at least one sentence in the language defined by G.
T **F** In an attribute grammar, a node's synthesized attributes can be based on the values associated with that node's children.
T **F** In most programming languages that have infix operators, +, -, * and / are right associative.
T F A recursive descent parser for a language can not be based directly on a grammar that has direct or indirect left recursion.
T **F** Most modern programming languages use both dynamic and static scoping for variables.
T F A variable's lifetime is the time during which the variable is bound to a storage location in memory.
T F An LR parser always scans the input from left to right and produces a rightmost derivation if one exists for the input string.
T F The idea behind operational semantics is to define the meaning of statements in a programming language by translating them into statements in another language.
T F Axiomatic semantics is an approach that is often used to prove that programs correctly implement their requirements.

T F The law of consequences says that if statement S has precondition P and post-condition Q, we can replace P with any P' that implies P and replace Q with any Q' that is implied by Q.

T **F** In general, for a grammar to be a good fit for a bottom up parser like an LR parser, it must satisfy the pair-wise disjointness test.

T F Every regular expression can be expressed as a deterministic finite automaton (also known as a deterministic finite state machine).

2. Match languages to domains (5 pts)

Write the letter corresponding with the application domain each programming language was designed to support in the space provided. Each letter should be used exactly once.

__**B**__ C	(A) Artificial Intelligence
__**E**__ COBOL	(B) System programming
__**D**__ FORTRAN	(C) Internet programming
__**A**__ LISP	(D) Scientific applications
__**C**__ JAVA	(E) Business applications

3. Attributes of implementation techniques (15 pts)

Here are four techniques that can be used to implement a programming language. For each one, briefly describe the significant attributes that strongly favor or disfavor the approach. You might consider attributes such as debugging ease, execution speed, translation speed, compactness, portability, etc.

(a) direct execution on hardware
(b) compilation to a low level language such as a machine language
(c) compilation to an intermediate language (e.g., "byte code") that is then interpreted in software
(d) direct interpretation in software

(a) is an approach seldom used for higher level programming languages. To some degree, it has been used for some specialized languages such as Lisp and Prolog. This approach would be expected to give the fastest execution speed as well as no translation speed and a compact representation (i.e., the source code itself). Weak points will be debugging and portability – the language would run only on the given hardware. (b) is the approach taken by languages like C, which is compiled down to machine language. The execution speed should be very fast, but the translation time might be long, the portability of the translation (the object code) will not be portable, and debugging the executable will be primitive. (c) is the approach used for many languages today, including Java. Execution speed is ok, but not as good as in (b). Object code size is reasonable – between source code and machine language. Debugging is probably not so good. Portability is a big win. (d) is the approach traditionally used in languages like Lisp, Prolog and Basic. Translation time is minimal, compactness is high, portability is high and debugging ease is very good. The only negative is execution speed.

4. Static and dynamic scope (15 pts)

Consider the program in the box to the right. Assuming dynamic scoping, what is the output of:

Statement 1: _____1,2_____

Statement 2: _____1,200_____

Statement 3: _____10,200_____

Statement 4: _____1,200_____

Assuming static scoping, what is the output of:

Statement 1: _____1,2_____

Statement 2: _____1,200_____

Statement 3: _____10,200_____

Statement 4: _____1,200_____

```
program main;

  var x, y : integer;

  procedure sub1;
    var y : integer;

    procedure sub2;
       begin
         print x, y;  //statement 1
       end; // sub2

    begin
      x = 1;
      y = 2;
      sub2;
    end;  // sub1

  procedure sub3;
    var x : integer;
    begin
      x = 10;
      print x, y;  //statement 3
    end;  // sub3

 begin
   x = 100;
   y = 200;
   sub1;
   print x, y;  //statement 2
   sub3;
   print x, y;  //statement 4
 end:  // main
```

5. Ambiguous grammars (15 pts)

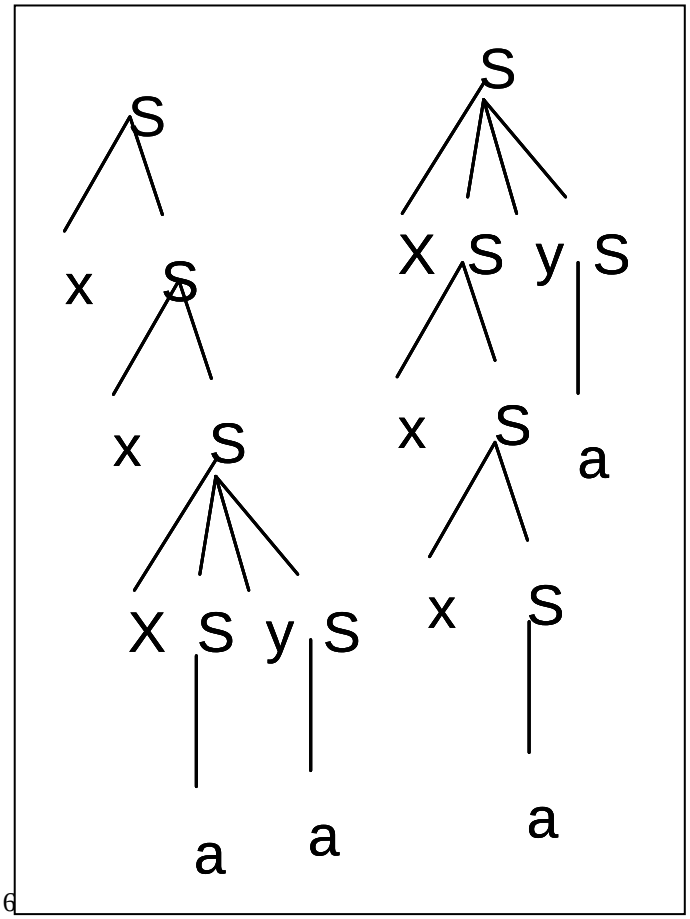
Which of the following grammars G1 and G2 is ambiguous.

```
G1: <S> ::= x <S> z | x <S> y <S> z | a
G2: <S> ::= x <S> | x <S> y <S> | a
```

(a) (5 pts) Which of these two grammars is ambiguous:
G1 **G2** *(circle one)*

(b) (10 pts) Show that this grammar is ambiguous by giving two different parse trees for a single sentence in the language defined by the grammar.

The sentence xxxaya has two parse trees. As the following diagram shows.



6. Operators (20 pts)

Given the following BNF grammar for a language with two infix operators represented by % and @ .

```
<foo> ::= <bar> @ <foo> | <bar>
<bar> ::= <mumble> | <bar > % <mumble>
<mumble> ::= ( <foo> ) | a | b | c
```

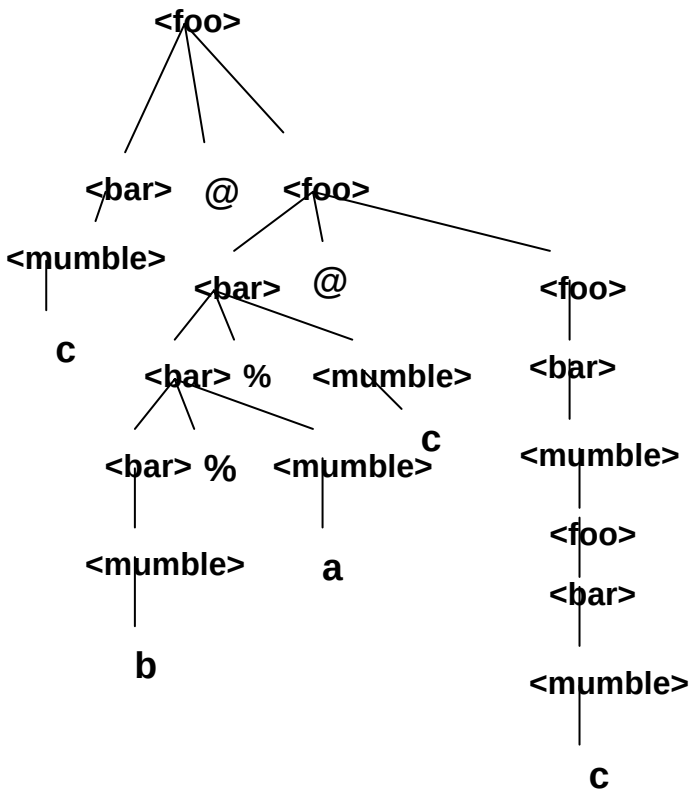
A. (5 pts) Give a parse tree for the following string:

```
c @ b % a % c @  ( b )
```

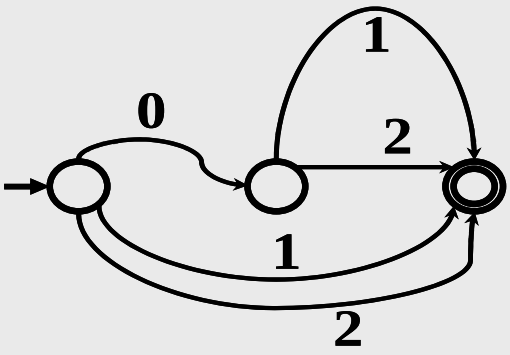
B. (5 pts) Circle the associativity of the @ operator: left **right** neither

C. (5 pts) Circle the associativity of the % operator: **left** right neither

D. (5 pts) Which operator has higher precedence: **%** @ neither



7. Specifying a language (30 pts)



This example shows a DFA for the RE **`0?[12]`** which might also be described using the CFG:

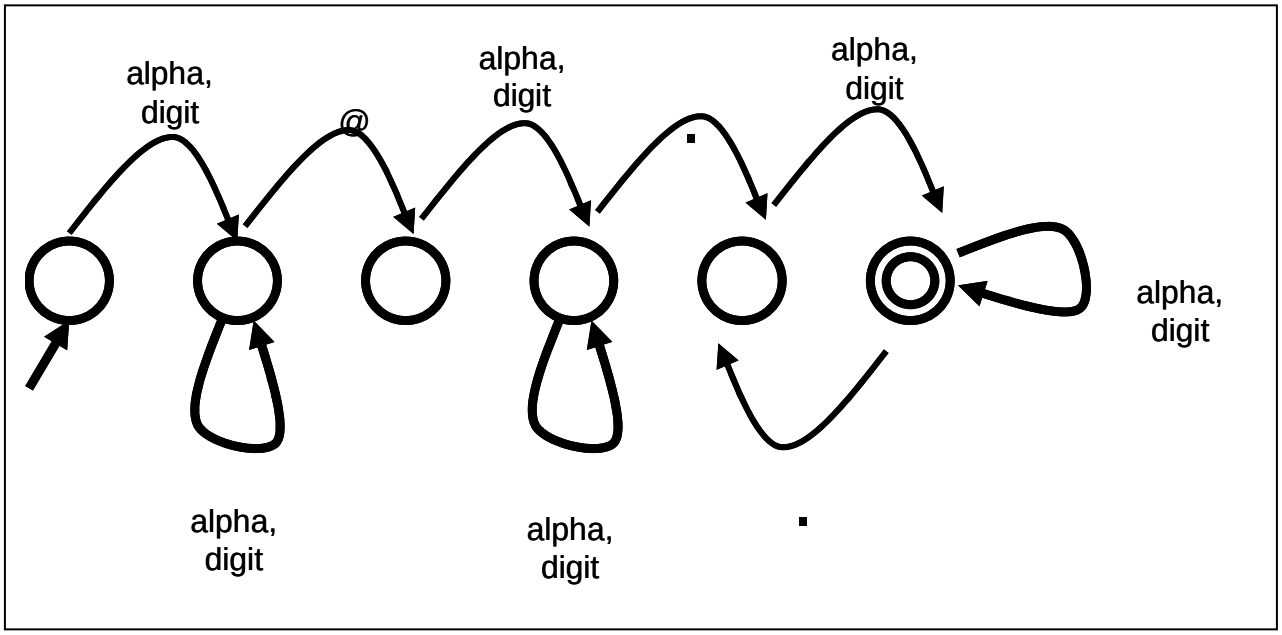
`S -> 0A | A  A -> 1 | 2`

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 alpha-numeric 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) (10 pts) 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) (10 pts) 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) (10 pts) 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.



RE: (alpha | digit)+ @ (alpha | digit)+ . ((alpha | digit)+)+

CFG:

```
Email -> Username @ Host
Username -> Alphanums
Host -> Alphanums . Alphanums | Alphanums . Host
Alphanums -> Alphanum | Alphanum Alphanums
Alphanum -> alpha | digit
```

8. LR parsing (30 pts)

Consider an LR parser for the following simple expression grammar, producing the table to the right.

```
1: E -> E+T
2: E -> T
3: T -> T*F
4: T -> F
5: F -> (E)
6: F -> id
```

	Action						Goto		
State	id	+	*	(	)	$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

(a) (1pt) How many states are there? **eleven**

(b) (1pt) What does the $ represent? **End of input**

(c) (1pt) In an action **S5**, what does the 5 represent? **Shift and go to state five**

(d) (1pt) In the action **R2**, what does the 2 represent? **Reduce using the second grammar rule**

(e) (1pt) What does the action **accept** represent? **The input is in the language and parsed ok**

(f) (15 pts) Complete the following table which shows the first five steps of an LR parser parsing the input string **id*id*id**

STACK	INPUT	ACTION
0	id*id*id$	Shift 5
0 id 5	*id*id$	Reduce 6, use goto(0,F)
0 F 3	*id*id$	Reduce 4, use goto(0,T)
0 T 2	*id*id$	Shift 7
0 T 2 * 7	id*id$	Shift 5

(g) (10 pts) For the sentential form **T*F*id ,** give the phrases, simple phrases and handle.

This sentential form corresponds to the parse tree: (E (T (T (T * F)) * (F (id))))

Phrases: **T*F*id, T*F, id**
Simple phrases: **T*F, id**
Handle: **T*F**