1	40/
2	20/
3	15/
4	30/
5	30/

	145/

CMSC 331 Midterm Exam, Fall 2011

Name: _____

UMBC username: _____

You will have seventy-five (75) minutes to complete this closed book/notes 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.

1. True/False [40]

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

T F **1.1** PASCAL was designed for teaching computer programming. **T**
T F **1.2** FORTRAN is an example of the *procedural* programming paradigm. **T**
T F **1.3** The use of a "virtual machine" for a programming language combines aspects of an interpreter and compiler. **T**
T F **1.4** A programming language feature is called *syntactic sugar* if it allows one to express programs in the language that cannot be expressed without it. **F**
T F **1.5** Any finite language can be defined by a context free grammar. **T**
T F **1.6** No symbol can be both a terminal and non-terminal symbol in the given grammar. **T**
T F **1.7** A recursive descent parser cannot directly use a grammar that has both left and right recursive rules. **T**
T F **1.8** Lexical scanners are usually defined with regular expressions. **T**
T F **1.9** Every non-deterministic finite automaton can be re-written as a deterministic finite automaton. **T**
T F **1.10** In an unambiguous grammar for a language, there is only one derivation for every string in the language. **F**
T F **1.11** Non-terminal symbols can appear on either the left or right hand side of a BNF rule. **T**
T F **1.12** When matching patterns from input, Lex uses the pattern that consumes the most characters. **T**
T F **1.13** Yacc parses using an LR(1) type grammar. **T**
T F **1.14** Axiomatic semantics associates pre-conditions and post-conditions with each statement in a program. **T**
T F **1.15** While a sentence in a language can only contain *terminal* symbols, a sentential form can also contain one or more *non-terminal* symbols. **T**
T F **1.16** A lexical scanner reads a stream of tokens and produces a parse tree. **F**
T F **1.17** BNF grammars and context free grammars are essentially equivalent. **T**
T F **1.18** Orthogonality in a programming language refers to the property that language features can be combined freely. **T**
T F **1.19** A bottom up parsing approach can not be used with some grammars. **F**
T F **1.20** All current programming languages can be defined by context free or BNF grammars. **F**

2. Programming language implementation techniques [20]

Here are four techniques that can be used to implement a programming language. For each one, briefly describe in a sentence or two 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., the Java Virtual Machine) 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

3. Axiomatic Semantics [15]

Show the weakest precondition (e.g., the **{ ?? }**) for the following statements and postcondition.

(a) **{ ?? }** x = 2y; { x>12 } *(Hint: be careful here; the two x's are different. Think about what **must** be true before the statement given that x>21 is true after.*

{y > 6}

(b) **{ ?? }** x = 2*y; y = 2*x; { y>16 } *(Hint: first compute the weakest precondition that holds after the first statement is done and before the second executes. Then back that up to compute the weakest precondition that holds before the first is done.)*

Between the two statements {X>8} is true, so the precondition before the sequence is {y>4}.

(c) { ?? } if (x>0) x = x + 1; else x = x-1; { x > 2 } *(Hint: consider the two possible cases that determine which branch of the conditional expression is taken, one where the condition x>0 holds and one where it does not..)*

{x > 1}

4. Simple grammars [30]

Here's a grammar for logical sentences with the start symbol s. The symbols and, or and not refer to the familiar concepts in logic: conjunction, disjunction and negation.

s => c
c => c and d | d
d => a | a or d
a => not a | 0 | 1

(a) [2] What are the non-terminal symbols? **{s, c, d, a}**

(b) [2] What are the terminal symbols? **{0, 1, and, or, not}**

(c) [2] What are the operators? **{and, or, not}**

(d) [2] Which operator has the highest precedence? **not**

(e) [2] Which operator has the lowest precedence? **and**

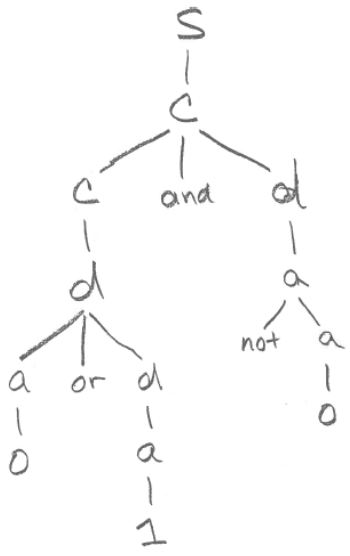
(f) [2] What is the associativity of the 'and' operator: (i) left associative; (ii) right associative; (ii) neither left nor right associative. **i: left associative**

(g) [3] Is the string "not not 0 and 1" in the language defined by this grammar? **yes**

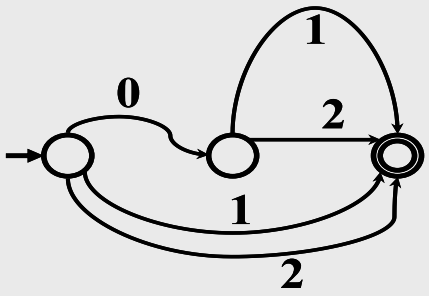
(h) [3] Is the string "not 0 not not 1" in the language defined by the grammar? **no**

(i) [2] Is the grammar ambiguous? **no**

(j) [10] Show all of the parse trees that can be generated for the string "0 or 1 and not 0"



5. Regular expressions [30]

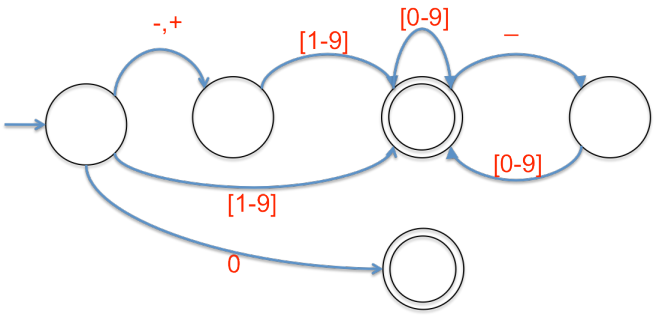


This example shows a DFA for the regular expression `0?[12]`

In the Ada, an integer constant contains one or more digits, but it may also contain embedded underscores. Any underscores must be preceded and followed by at least one digit, which implies that a number may not begin or end with an underscore, and numbers may not contain two or more adjacent underscores. Underscores have no significance other than readability. You propose that in Ada-2012, integer constants can be preceded by an optional sign (+ or -) and may not start with a "leading" 0, i.e. one followed by an underscore or more digits. Moreover, the integer 0 may not be preceded by a sign. Example of legal Ada-2012 integer constants are : 0, -17, 17, 1_001_7, +1_048_576. Illegal examples include 01, 0_34, 14_, 1__3, +_12, -0 and +0.

(a) [15] Draw a DFA (Deterministic Finite Automata) that recognizes Ada integer. Minimize your DFA to emerge any duplicate states (i.e., states that are both accepting state or both non-accepting states and have arcs labeled with the same input symbols that transition to duplicate states). Feel free to define a class of characters using a notation like the following, which represents a letter and a single digit and to put such a class name on an arc in your DFA.

```
LET: [a-zA-Z]
DIG: [0-9]
```



(b) [15] Write a regular expression to recognize an Ada integer. Use a notation in which a '*' indicates any number of repetitions, '+' indicates one or more repetitions, '?' means zero or one repetitions, parentheses group things, a vertical bar separates alternatives, etc., as in the following example for names of people.

```
LET: [a-zA-Z]
((mr|mrs|ms|dr)\.\t+)? {LET}+ (\t+ {LET}+)*
```

(0 | ([\-+]? [1-9] ([0-9]* | _ [0-9])*)