1	40/
2	15/
3	45/
4	35/
5	30/
6	20/
7	15/

	200/

CMSC 331 Midterm Exam Fall 2010

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 COBOL is considered by many to be the first object-oriented programming language. **FALSE**
T F The *imperative* and *procedural* programming paradigms are names for the same thing. **TRUE**
T F The "Von Neumann" computer architecture is characterized by hardware that has a CPU separate from memory, which is used to hold both data and programs. **TRUE**
T F One of the advantages of interpreted languages is their faster execution time compared to compiled languages. **FALSE**
T F A finite language is one that consists of a finite number of sentences. Hence, no finite language can be defined with an ambiguous grammar. **FALSE**
T F Lexical scanners are usually defined with attribute grammars. **FALSE**
T F A recursive descent parser is a simple, yet popular kind of bottom-up parser. **FALSE**
T F Every regular expression can be transformed into an equivalent deterministic finite automaton (also known as a deterministic finite state machine). **TRUE**
T F A non-deterministic finite automaton for a regular language is generally easier to write than a deterministic one, but harder to apply to a string to see if it matches. **TRUE**
T F If the grammar for a language is ambiguous, then some valid sentence in that language has more than one parse tree. **TRUE**
T F BNF grammars do not allow left-recursive rules. **FALSE**
T F BNF grammars can be used for both language generation and language recognition. **TRUE**
T F If two languages have different non-terminal symbols and productions, the languages cannot be identical. **FALSE**
T F Associativity rules only apply to operators of the same precedence level. **TRUE**
T F Attribute grammars can capture features in a language that BNF grammars cannot. **TRUE**
T F A grammar with left recursive rules cannot be directly used to implement a recursive descent parser. **TRUE**
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. **TRUE**
T F A special form in Scheme is an expression that is not evaluated, i.e., it represents literal data. **FALSE**
T F When evaluated, a lambda expression always returns a function. **TRUE**
T F A closure in Scheme is just a function. **TRUE**

2. General multiple-choice questions [15]

Circle all of the correct answers and only the correct answers.

2.1 Which of the following is not considered a functional programming language? (a) ML; (b) Haskell; (c) Smalltalk; (d) Scheme; (e) Lisp. (f) Java (g) Algol **(C, F, G)**

2.2 *Left factoring* is a technique that is used to (a) produce a left most derivation of a string from a grammar; (b) remove left recursion from a grammar; (c) remove right recursion from a grammar; (d) eliminate a non-terminal from the left side of a grammar rule; (e) none of the above **(B)**

2.3 A LR(1) parser (a) processes the input symbols from left to right; (b) produces a left-most derivation; (c) looks ahead at most one input symbol before knowing what action to take; (d) takes time proportional to the cube of the number of input symbols **(A, C)**

2.3 Which of the following Scheme expressions would be interpreted as true when evaluated: (a) 0 (b) 1 (c) empty (d) #t (e) (lambda () #f) (f) #f **(A,B,C,D,E)**

2.4 In Scheme, a lambda expression represents a (a) abstract class; (b) variable type; (c) function; (d) conditional; (e) cons cell. **(C)**

3. Operators [45]

Consider the language defined by this BNF grammar where <CLAUSE> is the initial symbol.

```
<UNIT>    ::=  ( <CLAUSE> ) | a | b | c
<ITEM>    ::=  not <UNIT> | <UNIT>
<CLAUSE>  ::=  <CLAUSE> and <PHRASE> | <PHRASE>
<PHRASE>  ::=  <ITEM> | <ITEM> or <PHRASE>
```

(a) [5] What are the terminal symbols? **{ (,), a, b, c, and, or, not }**

(b) [5] What are the non-terminal symbols? **{<CLAUSE>, <PHRASE>, <UNIT>, <ITEM>}**

(c) [5] Circle the associativity of the 'and' operator: (i) left; (ii) right; (iii) neither **LEFT**

(d) [5] Circle the associativity of the 'or' operator: (i) left; (ii) right; (iii) neither **RIGHT**

(e) [5] Circle the associativity of the 'not' operator: (i) left; (ii) right; (iii) neither **NEITHER**

(f) [5] Which operator has highest precedence: (i) and; (ii) or; (iii) not; (iv) none of the above **NOT**

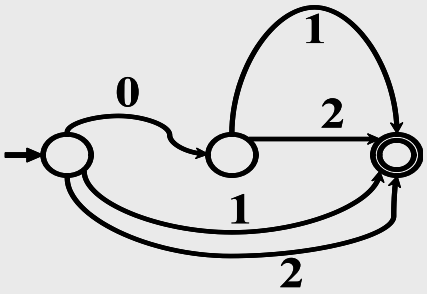
(g) [15] Draw a parse tree for the following string:

c and b or not a and (not b or c)

Here is the parse tree in a lispy notation.

```
(clause (clause (clause (phrase (item (unit 'c'))))
                'and'
                (phrase (item (unit 'b'))
                        'or'
                        (phrase (item 'not'
                                      (unit 'a')))))
        'and'
        (phrase (item (unit '('
                            (clause (phrase (item 'not'(unit 'b'))
                                            'or'
                                            (phrase (item (unit 'c')))))
                            ')' ))))
```

4. Regular expressions [35]



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

Define a language for simplified email addresses using a finite automaton (either deterministic or non-deterministic) and a regular expression (RE). Use the normal conventions for FAs and REs shown in the example to the right. An email address has:

- An account name starting with a letter ([a-zA-Z]) and continuing with any number of letters or digits ([0-9])
- An @ character
- A host with two or more sequences of letters or digits separated by periods. The last sequence must be a 'toplevel domain, i.e. either 'edu', 'org', or 'com'.

Here are examples of strings that should be in and out of the language:

IN: x@y.org, foo@bar.com, x1z@100.bb.cc.edu
OUT: 1st@foo.com, ab@x.y, a@edu, b@.umbc.edu, me@aa.b.cc.d.e.ff.g

(a) [15] Draw a DFA for this language. 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.

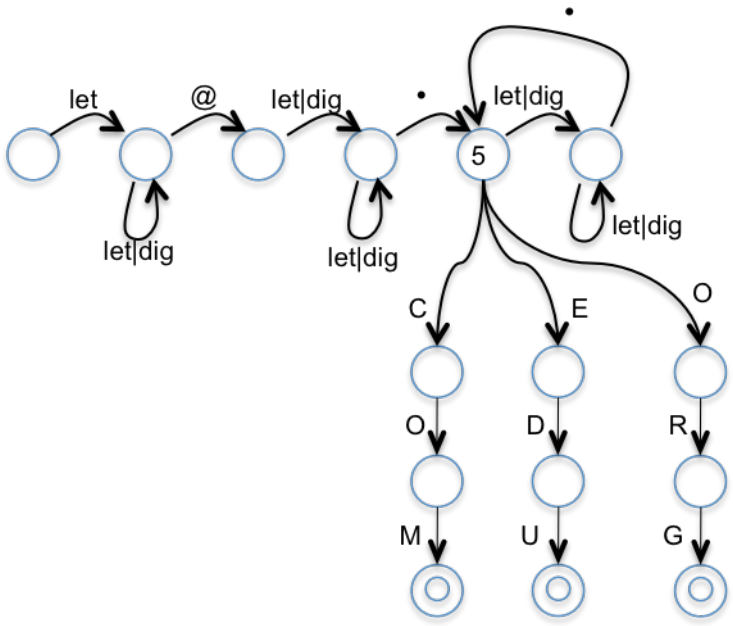
```
LET: [a-zA-Z]
DIG: [0-9]
```

(b) [5] Is your FA deterministic or non-deterministic? **Non-deterministic**

(c) [15] Write an equivalent regular expression for your DFA. Use a notation like the following, which describes a simple RE for names of people. Recall that '*' indicates any number of repetitions, a '+' indicates one or more repetitions, a '?' means zero or one repetitions, '\.' Is a period, and '\t' represents a white space character. Use parentheses to group things. A vertical bar separates alternatives.

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

```
LET: [a-zA-Z]
DIG: [0-9]
LET (LET|DIG)* @ ((LET|DIG)+\.)+(edu|com|org)
```



5. Constructing s-expressions [30]

Consider the Scheme data Structure that when printed looks like (((1) 2) 3)

5.1 [5] Give a Scheme expression using only the **cons** function that will create this list. Use the variable **empty** for the empty list.

```
(cons (cons (cons 1 empty) (cons 2 empty)) (cons 3 empty))
```

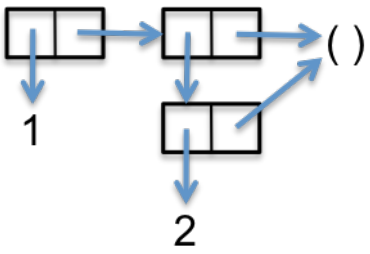
5.2 [5] Give a Scheme expression using only the **list** function that will create this list.

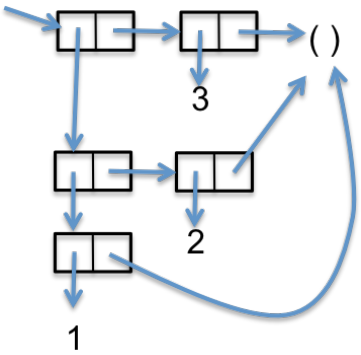
```
(list (list (list 1) 2) 3)
```

5.3 [10] Assuming that we've done **(define x '(((1) 2) 3))** give Scheme expression using only the functions **first** and **rest** and the variable **x** that returns each of the three symbols in the list.

symbol	*s-expression to return the symbol*
1	**(first (first (first x)))**
2	**(first (rest (first x)))**
3	**(first (rest x))**

5.4 [10] Draw a "box and pointer" diagram showing how the list (((1) 2) 3) is represented in pairs. The figure to the right shows an example of the diagram format you should use. This example represents the list (1 (2)).





6. Scheme I [20]

Your roommate is obsessed with the functional programming paradigm and wants to use it to define *mylength*, a procedure that takes a list as a parameter and returns the number of top-level elements in it. Her approach is to first replace every element in the list with 1, and then use *sumlist* to add them up. She already defined *sumlist* as

```
(define (sumlist lst)
    (if (null? lst) 0 (+ (first lst) (sumlist (rest lst)))))
```

She explains that she will compute the length of the list (3 2 foo (3 4)) by constructing the list (1 1 1 1) and applying sumlist to it to get 4.

(a) [10] She started to define his length procedure below, but couldn't figure out what to put in the space where the box is. Finish the definition by providing the missing Scheme expression.

```
(define (mylength lst)
  (sumlist
    (map    (lambda (x) 1)  lst)))
```

(b) [10] Show her how to be even more functional by using the built-in Scheme function *reduce* to eliminate the need for *sumlist*. The definition of *reduce* is given in the figure to the right. Complete the following definition by providing the missing expression for each line.

```
(define (reduce f final lst)
  (if (null? lst)
      final
      (f (first lst)
         (reduce f
                 final
                 (rest lst)))))
```

```
(define (mylength lst)

  (reduce +  0  (map (lambda (x) 1) lst) ))
```

7. Scheme II [15]

Consider the following mystery function.

```
(define (xyzzy l1 l2)
  ;; xyzzy takes two lists and returns...
  (cond ((null? l1) empty)
        ((null? l2) empty)
        ((equal? (car l1) (car l2))
         (cons (car l1) (xyzzy (cdr l1) (cdr l2))))
        (else empty)))
```

(a) [5] Explain what the function does in a sentence or two.

it returns the longest common prefix of the two lists

(b) [5] Give three examples of a call to the xyzzy function and what it would return that illustrate its behavior,

```
> (xyzzy '(1 2 3 4) '(1 2))
(1 2)
> (xyzzy '(1 2 3 4) '())
()
> (xyzzy '(1 2 3 4) '(1 2 3 4 5 6 7))
(1 2 3 4)
> (xyzzy '(a () 1 c) '(a (1) 1 c))
(a)
> (xyzzy '(a b c d) '(1 2 3 4))
()
```

(c) [5] What would be a good name for the function

longest-common-prefix