1		/ 40
2		/ 20
3		/ 10
4		/ 50
5		/ 20
6		/ 20
7		/15
8		/10
9		/ 15
10		/ 15
11		/ 15
total		/ 230

UMBC CMSC 331 Final Exam
11 December, 2003

Name:_____StudentID#:_____

You will have two hours to complete this closed book exam. We reserve the right to assign partial credit, and to deduct points for answers that are needlessly wordy or simply wrong.

1. General multiple-choice questions (40)

Circle the letter (e.g., **(c)**) for the single correct answer for each problem (4 points each).

1.1 Attribute grammars are used to specify (a) basic syntax of a programming language; (b) non-finite state machines; (c) static semantics of a programming language; (d) dynamic semantics of a programming language.

1.2 Assuming that y is of type String, what happens in the Java statement ``String x = y"? (a) The wrapper method for a String is called to create a new string object and the result bound to x; (b) The address of y is modified to be the address of x. (c) The object bound to y is copied and bound to x, and any previous binding of x to an object is lost. (d) x and y become aliases. (e) the contents of y are copied into the storage allocated to x.

1.3 Which of the following is true of EBNF notations (extended BNF): (a) EBNF notations allow one to describe the semantics of a programming language; (b) any grammar in a EBNF notation can be also be expressed in a regular BNF notation; (c) EBNF notations allow one to describe some grammars that can not be described by regular BNF notations; (d) EBNF can encode attribute grammars where as regular BNF notations can not.

1.4 What best distinguishes a purely ``functional" programming language from an ``imperative" one? (a) There are no variables and hence no assignment operation in a purely functional language. (b) A purely functional language lacks the ``go to" statement, but an imperative language always has such a command. (c) All subprograms must be declared with the keyword function in a purely functional language. (d) There is no real difference, only a difference in the recommended coding style.

1.5 The main difference between a sentence and a sentential form is (a) there is no difference; (b) a sentence contains only terminal symbols but a sentential form can contain some non-terminal symbols; (c) sentential forms are a subset of sentences but the converse is not true; (d) sentential forms have no handles but a sentence does.

1.6 The language that is generally regarded as the first object-oriented language is (a) Java; (b) C++; (c) Simula; (d) Lisp; (e) COBJOL; (f) Ada; (g) Logo; (h) PL/I.

1.7 The Unix YACC utility program parses input using a (a) recursive descent parser; (b) a top-down parser; (c) an RR(1) parser; (d) a bottom-up shift reduce parser.

1.8 Axiomatic semantics is often used for (a) program verification; (b) byte code generation; (c) syntax-directed parsing; (d) encoding an unambiguous language specification.

1.9 Operator associativity generally applies to (a) only prefix operators; (b) only infix operators; (c) only postfix operators; (d) prefix, infix and postfix operators.

1.10 A tail-recursive algorithm is generally better than a non-tail recursive algorithm because (a) it can be run without growing the stack; (b) it is easier to understand; (c) it has no side-effects; (d) all of the above.

2. Regular expressions (20)

The following regular expression recognizes certain strings consisting of the letters A, B and C.

A ((AB) | (AC))* C

Recall that the *Kleene star* (*) means zero or more repetitions of the pattern it follows, the vertical bar is an infix operator delimiting alternatives and parentheses are used for grouping patterns. For each of the following strings, circle the response to show whether it is IN or OUT of the language defined by this regular expression (3 points each)

2.1 [IN] [OUT] AACC
2.2 [IN] [OUT] ABAC
2.3 [IN] [OUT] AC
2.4 [IN] [OUT] ABABABABACAC
2.5 [IN] [OUT] AABACC

2.6 Draw a deterministic finite state machine (also known as a deterministic FSM or a deterministic finite automaton or DFA) for the language defined by this grammar. Label each arc with the input required for that transition. Mark the initial state with an arc coming into the state and mark all final states with a double circle. (5 points)

3. Ambiguous grammars (10)

Briefly describe (3.1) what it means for a grammar to be ambiguous and (3.2) how one can prove that a grammar is ambiguous. (5 points each)

4. Java true/false questions. (50)

Mark each assertion as true or false by circling [T] or [F]. (2 points each)

[T] or [F] : The JRE is a package for running Java programs and JDK for compiling Java programs.

[T] or [F] : A class that is abstract may not be sub-classed.

[T] or [F] : The value of a final Java variable cannot be changed once it has been initialized.

[T] or [F] : Every Java application has a **main** method that is **static.**

[T] or [F] : A change in one Java class may cause a syntax error in a different Java class.

[T] or [F] : A subclass can access `private` variables in its superclass.

[T] or [F] : In the Model-View-Controller design pattern, the user interface belongs in the Model component.

[T] or [F] : In Java, every if statement must have both a then clause and an else clause.

[T] or [F] : Like Lisp, Java uses garbage collection to perform memory management.

[T] or [F] : If one Java method overloads another they can not have identical signatures.

[T] or [F] : If one Java method overrides another they can not have identical signatures.

[T] or [F] : The java package system allows you to group classes in subdirectories to avoid accidental name conflicts.

[T] or [F] : Static methods do not require an instance of the class; they can be accessed through the class name.

[T] or [F] : A single java try statement can have more than one catch statement.

[T] or [F] : Java strings are immutable objects.

[T] or [F] : Within a single Java Virtual Machine (JVM), all runable threads share the same run-time stack.

[T] or [F] : When a thread is waiting for an I/O operation, it is said to be blocked.

[T] or [F] : The Java Virtual Machine compiles Java byte code into machine language instructions.

[T] or [F] : Java does not have true multiple inheritance like C++.

[T] or [F] : A Java file must define exactly one non-anonymous class.

[T] or [F] : Since arrays are objects in Java they inherit all the characteristics of java.lang.Object.

[T] or [F] : A Java string is not a primitive data type but rather is defined as an object.

[T] or [F] : Swing was the original Java GUI toolkit and the AWT was added in Java 2.

[T] or [F] : A wrapper is an object class that corresponds to a primitive data type like int.

[T] or [F] : Downcasting is generally required when your program takes instances out of a vector or arrayList..

5. Constructing s-expressions (20)

In the following table, the first column lists some s-expressions we would like to construct. Complete the rest of the table by giving the simplest possible expression to build the s-expression. The expression in the second column should only use the CONS function and the one in the third should only use LIST. If it's not possible to construct an s-expression say so. You are not permitted to include a quoted list (like '(b c)) in giving you answer. You are permitted, of course to use quoted atoms (e.g., ‘A) and also to use NIL (which need never be quoted). We've done the first row for you as an example. (2 points each)

	SEXPs	**Using CONS**	**Using LIST**
--	*(a)*	*(cons ‘a nil)*	*(list ‘a)*
7.1	**(a b)**		
7.2	**((a))**		
7.3	**(a ((b) c))**		
7.4	**(a (b . c))**		
7.5	**(( ) (( )))**		

6. Deconstructing S-expressions (20)

In the following table, the first column lists some s-expressions containing a single instance of the atom X. Assume that the variable S is bound to the s-expression in the first column. The second column should be an expression that when evaluated returns the atom X. You are only allowed to use the functions CAR and CDR and a single instance of the variable S. We've done the first row for you as an example. (4 points each)

	If S is this	**Expression to return atom x**
--	*(x)*	*(car s)*
8.1	(a b x)	
8.2	(((x)))	
8.3	(a (b (c x)))	
8.4	(a (b (c)) x)	
8.5	(a (b . x))	

7. Writing a Lisp function (15)

Consider a function **`insert-at`** `that` takes 3 arguments: an *element* (which could be a LISP atom or list), a *list*, and a positive *integer*. The function should return a new list that is the result of inserting the first argument into the second argument at the position specified by the third argument. Note that positions begin with zero. For example,

```
> (insert-at 'foobar '(a b c d) 3)
(A B C FOOBAR D)
> (insert-at '(foobar) '(a b c d) 1)
(A (FOOBAR) B C D)
> insert-at ‘foobar ‘(a b c) 6)
(a b c foobar)
> insert-at ‘foobar ‘(a b c) 0)
(FOOBAR A B C)
```

Here is an incomplete definition of the function. Give code expressions for <*SEXP1*>, <*SEXP2*> and <*SEXP2*> that will complete it.

```
(defun insert-at (element list position)
    (if <SEXP1>
        (cons element list)
        (if (null list)
            <SEXP2>
            (cons (car list) <SEXP3>))))
```

<SEXP1>	
<SEXP2>	
<SEXP3>	

8. Lambda expressions (10)

To what do the following expressions evaluate? For example ((lambda (x y)(+ x y)) 3 4) would return 7. (worth 2, 4, 4 points)

8.1 ((lambda nil 33))

8.2 ((lambda (x y)(+ x y)) 44 ((lambda (x y)(+ x y)) 22 33))

8.3 (funcall (lambda (x y z) (* z (+ x y))) 3 5 7)

9. Functional programming I (15)

One aspect of function programming is the use of functions which take other functions as arguments. The Lisp mapcar function is a classic example. What do the following expressions return? Each is worth 5 points.

9.1 (mapcar #'integerp '(1 a 3 b))

9.2 (mapcar (function +) (mapcar #'1+ '(1 2 3)) '(4 5 6))

9.3 (mapcar (function cons) '(1 2 3) '(a (b) ((c))))

10. Functional programming II (15)

Suppose we defined a function MAPCDR as follows:

```
(defun mapcdr (f l)
     (if (null l)
          NIL
          (cons (funcall f l) (mapcdr f (cdr l)))))
```

What would the following expressions return? If you believe that the expression would cause an error, say so. Each is worth 5 points.

10.1 (mapcdr #'length '(a b c d))

10.1 (mapcdr #'car '(a b c d))

10.1 (mapcdr #'(lambda (x) x) '(a b c d))

11. Functional programming III (15)

Another aspect of functional programming is the ability to write functions to create new functions. Consider the function ZYZZY defined as follows:

```
(defun xyzzy(f)(lambda(x)(not(funcall f x))))
```

Note: NOT is a built-in that could be defined as (DEFUN NOT (X) (IF (NULL X) T NIL))

11.1 With what type of argument should xyzzy be call? (2)

11.2 What type of thing does xyzzy return? (2)

11.3 Describe in a sentence what xyzzy does? (6)

11.4 What will (mapcar (xyzzy #'evenp) '(1 2 3 4 5 6)) return? (5)

12. Have a good holiday break (priceless)