1	50/
2	40/
3	35/
4	10/
5	15/
6	50/
7	20/
8	25/
9	10/
10	15/
11	20/
12	10/
300/	

15 December 2008

CMSC 331 Final Exam Fall 2008

Name: _____

UMBC username:_____

You have two hours 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. Skim through the entire exam before beginning to get a sense of where best to spend your time. If you get stuck on one question, go on to another and return to the difficult question later. Comments are not required for programming questions but adding some might help us understand your code.

1. True/False (50 pts: 25*2)

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

T F BNF grammars do not allow left-recursive rules.

T F Lexical scanners are typically defined using context-sensitive grammars.

T F A language defined by a regular expression can always be defined using a BNF grammar.

T F One disadvantage of static type checking is that it incurs extra type checking during run-time, slowing down the execution speed of most programs slightly.

T F In most programming languages that have infix operators, the standard arithmetic operators (e.g., +, -, * and /) are left associative.

T F Scheme does not allow you to assign a variable more than once.

T F Scheme interpreters execute tail-recursion without growing the stack.

T F Lazy evaluation is a technique that can make it easy to avoid unnecessary computation.

T F A continuation in Scheme is another name for a function closure.

T F Scheme functions look up the values of non-local variables in the environment of their caller.

T F Scheme's call-with-current-continuation allows one to more easily define functions that can do backtracking.

T F The Scheme-in-scheme interpreter does not really implement a lexical scanner or a parser.

T F The Scheme-in-python interpreter uses a recursive descent parser.

T F Trampolining is a way to implement recursion using iteration.

T F In Python, any value that is not considered as the False is interpreted as True.

T F Python allows you to define a class with more than one super class and inherits attributed and methods from all of them.

T F In Python, every function always returns a value.

T F In a Python dictionary, all of the keys must be unique.

T F In Python, a sting is just a list of characters.

T F In Python, the only difference between a tuple and a list is that a tuple's length is fixed.

T F Only immutable data structures can be used as keys in a Python dictionary.

T F In languages with dynamic typing, any type errors are in general only detected at run time.

T F Languages with static typing do not allow user-defined types.

T F Only languages with static typing can be type safe.

T F Python 3.0 is unusual in that it is not backward compatible with earlier version of Python.

2. General multiple-choice questions (40 pts: 10*4)

Circle the letters for **all** of the correct answers, as in the following sample question.

0. *What programming languages did we study this semester? (a) Prolog; (b) Scheme; (c) Haskell; (d) OCaml; (e) Python; (f) Perl*

1. Which of the following Python expressions would be interpreted as false when evaluated: (a) False; (b) -1; (c) 0; (d) []; (e) ‘’; (f) {}; (g) false; (h) not(-1) .

2. Which phrase best describes the variable scoping used in Python? (a) wide scoping; (b) narrow scoping; (c) lexical scoping; (d) dynamic scoping; (e) structured scoping; (f) denotational scoping; (g) none of the above.

3. In the Scheme interpreters we studied in class, which of the following items are a part of the representation of a procedure definition**:** (a) the procedure’s name; (b) the procedure’s parameter names; (c) the procedure’s code; (d) the environment in which the procedure was defined; (e) the procedure’s type; (f) storage for the procedure’s arguments.

4. In the Scheme interpreters we studied in class, the data structure used for an environment is essentially (a) a list of frames; (b) an instance of the Environment class; (c) a list of variable names and their values; (d) a frame of stacks; (e) a hash table or dictionary.

5. Scheme macros are primarily used to define: (a) dynamically scoped environments; (b) closures; (c) functions that don’t evaluate all of their arguments; (d) reflective programs.

6. While a recursive descent parser can be natural and efficient for parsing Scheme programs, one drawback in using it in Python is that: (a) It is difficult to transform it into a grammar with no left recursion; (b) Python has a limited stack and does not optimize tail-recursion as iteration; (c) the implementation in Python is obscure; (d) it is slow compared to using the trampolining style of programming.

7. Which of the following languages are generally considered to be type safe? (a) Scheme; (b) Java; (c) C; (d) C++; (e) Python.

8. Which of the following languages use dynamic typing? (a) Lisp; (b) Scheme; (c) Java; (d) JavaScript; (e) Python; (f) ML; (g) C.

9. Which of the following functions were not part of John McCarthy’s original description of Lisp? (a) CAR; (b) DEFINE; (c) LENGTH; (d) IF; (e) EQ; (f) SET; (g) QUOTE.

10. Which of the following are true of Scheme’s **delay** function? (a) it is a special form; (b) it is used to make a process sleep for a period of time; (c) it can be used to implement lazy evaluation; (d) it returns a closure; (e) it returns a continuation.

3. Writing a BNF grammar (35 pts: 20/15)

Assume that Boolean expressions are built up from the following symbols:

- Binary infix operators: *, +, =>, <=>
- Unary prefix operator: ~
- Variables: A, B, C
- Parentheses: (,)

The ~ operator has a highest precedence, followed by * and +, which have equal precedence. Operators => and <=> have the lowest precedence. All operators are all left associative. Parentheses are used to group expressions in the usual manner. Examples and non-examples are shown to the right.

(a) Write a BNF grammar for this language. Be sure it generates all Boolean expressions given above. *Hint: you probably want to define four non-terminal symbols, e.g., <exp>, <term>, <factor> and <var>.* (b) Using your grammar, draw a parse tree for the expression *A*B => A*.

Positive Examples

```
A
(~ A * ~ B) + (~ C =>
A)
~ A * ~ C
~ (A * ~ C) + B
(A * B <=> B * A)
(~ (A)) <=> ((A))
~ ~ A
```

Negative Examples

```
A ~ => B
A B
* A
) A + B (
()
*
```

4. Reserved names (10 pts)

Languages like Python, Java, and C++ have reserved words like "if", "else", and "class" that can't be used as names for variables or functions. The PL/I language has no reserved words and in it one can write statements like if = while + then(else). What are some advantages and/or disadvantages to having no reserved words?

5. Slices in Python (15 pts: 5, 10)

(a) In a few sentences, describe the general notion of Python's slice operation and what kinds of data structures it applies to. (b) In the following Python interactive session, show what would be printed for the missing values.

```
>>> s = "umbcCMSC331"
>>> s[2:3]


>>> s[2:]


>>> s[:3]


>>> s[:]


>>> s[-1:-3]


>>> s[:-1]


>>> s[1:1]
```

6. Regular expressions (30 pts, 5/20/25)

The table to the right gives the symbols Python uses in regular expressions.

(a) A variable name in Python must start with a letter or underscore and may be followed by any number of letters, digits, and underscores. Give a Python regular expression that matches a legal Python variable name.

Python RE symbols

^	the beginning of the line
$	the end of the line
+	one or more times
?	at most one time
*	zero or more time
(...)	a group
(?:...)	a noncapturing group
\t	a tab
\n	a newline character
{n}	n times
{n, m}	a range at least n and at most m
[...]	a character class
.	any character
\s	whitespace
\d	a number
\b	a word boundary

(b) Fill in the following table. For each row, assume that we evaluate the expression

```
mo = re.match(pattern,string)
```

Fill in the missing values for *Match?* (i.e., does the pattern matched the string, *mo.group()* (i.e., what part of the string matched the pattern) and *mo.group(1)* (i.e., which is the first of the pattern's match groups). Enter N/A in a cell if no answer is appropriate for it.

Pattern	String	Match?	mo.group()	mo.group(1)
*a**	*apple*	*Yes*	*a*	*N/A*
(ab*){1,2}	abaa{1,2}			
ab|(c?)	abcbb			
a(ab)*a	abababa			
1*(01)*0*	10110			
a.([bc]+)	abcbcbcbc			

(c) Write a Python regular expression to match a date string for any date from January 1, 1900 to December 31, 2099 using the familiar notation like 10/08/2002. Here is a specification:

> *A date is a month followed by a slash followed by a day followed by a slash followed by a year. A month is an integer between 1 and 12. If the integer is a single digit, it can be preceded by an optional zero (e.g., 01/20/2009). A day is an integer between 1 and 31 and, if a single digit, can be preceded by an optional zero (e.g., 12/01/2008). A year will be an integer between 1900 and 2099, inclusive.*

You can ignore the fact that some months have fewer than 31 days. Strings your expression should match include 01/01/1900, 1/1/1900, 12/31/2080, 1/01/2009, 02/30/1984, 1/23/1945, and 10/8/2002 and strings that shouldn't match include 00/01/1900, 1/1/1899, 13/01/2000, and 10/8/02. *Hint: it will probably help if you draw a finite state graph for the regular expression first.*

7. Reference semantics (20 pts: 10/10)

Python uses reference semantics for variable assignment rather than value semantics.

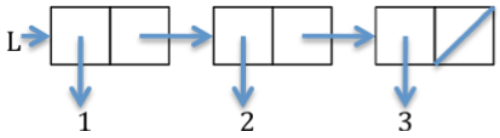
(a) Describe in a few sentences what reference semantics is and how I differ from value semantics.

(b) Fill in the values that Python would return in the following interactive session.

```
>>> L1 = [1,2,3,4]
>>> L3 = L2 = L1
>>> def foo(L1):
        L1 = [x+1 for x in L1]
        return L1
>>> foo(L2)


>>> L1


>>> L2


>>> L3


>>> L1 = foo(L2)
>>> L1


>>> L2


>>> L3
```

8. Destructive assignment in Scheme (25 pts, 5/5/5/5/5)

Scheme has two destructive assignment functions that can change the pointers in a cons cell: **set-car!** and **set-cdr!.** Assume each of the following six expressions are evaluated in order. After each is evaluated, show what will be printed for L and draw a box and pointer diagram of the structure of cons cells. Be sure to show any structure sharing.

Expression	Value of L	Diagram for L
`(define L '(1 2 3))`	`(1 2 3)`	
`(set-cdr! L (cdr (cdr L)))`		
`(set-cdr! L (cons 4 (cdr L)))`		
`(set-car! L (cdr L))`		
`(set-cdr! L (car L))`		
`(set-cdr! L L)`		

9. Explain this! (10 pts)

Your classmate wrote a simple function with two optional arguments: X which defaults to the string "a" and Y, which defaults to the empty list. It concatenates its first argument to itself and then appends that to the end of its second argument. Finally, it prints both arguments.

```
1    def foo(X = "a", Y = [ ]):
2        X = X + X
3        Y.append(X)
4        print X, Y
```

When she called foo( ) with no arguments repeatedly, she was surprised and puzzled by the results:

```
>>> for i in range(4): foo()
aa ['aa']
aa ['aa', 'aa']
aa ['aa', 'aa', 'aa']
aa ['aa', 'aa', 'aa', 'aa']
```

In a few sentences, explain to your classmate what is going on in this example and why the results are exactly as expected.

10. Explain that! (15 pts, 10/5)

Another classmate thinks he has found a bug in the Python interpreter. He shows you this code (see the box to the right) and expects bar1() and bar2() to print the same: the value of the global variable a and its double. But when he tried it out, he was puzzled.

```
>>> bar1()
100
200
>>> bar2()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in bar2
UnboundLocalError: local variable 'a' referenced
before assignment
```

```
a = 100

def bar1():
  print a
  print a + a

def bar2():
  print a
  a = a + a
  print a
  a = a / 2
```

(a) How would you explain to your classmate what Python is doing?

(b) How can he easily modify bar2() to have the effect he wants – to have bar2() produce the same output as bar1()?

11. Reversing strings and lists in Python (20 pts)

(a) Write a Python function rw() that takes a string representing a word and returns it reverse. *Hint: there are many ways to do this. One way is to produce a list of the string's characters, use the reverse method defined for lists, and join the results back into a string.*

(b) Now write a function rs() that takes a string representing a sentence and returns a string with the words in reverse order. *Hint: Split the string into a list of words, reverse the list, join them back together with a space between them*.

```
>>> rw('foo')
'oof'
>>> rw('')
''
>>> rs('help me')
'me help'
>>> rs('run')
'run'
>>> rs('the quick brown
fox')
'fox brown quick the'
```

12. Curry functions (10 pts)

Currying is the technique of transforming a function that takes multiple arguments so that it can be called as a chain of functions each with a single argument. Write a Python function *curry2* that takes as its argument a function of two arguments, and returns it in its curried form, i.e. a function of one argument that when called with X returns a function that when called with Y, returns the original function applied to the two arguments, X and Y. Se the sample session in the box to the right. *Hint: You'll use lambda expressions, of course.*

```
>>> def add(X,Y):
        return X+Y
>>> add(100,200)
300
>>> f = curry2(add)
>>> f(100)(200)
300
```