# CMSC 331 Final Exam Spring 2011

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 (25 points)

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

T F 1. Scheme uses static scope whereas ordinary Lisp uses dynamic scope.

T F 2. In Perl, a variable's type can change during the course of execution.

T F 3. If a Perl function terminates without explicitly returning a value, it returns the value nil.

T F 4. A disadvantage of dynamic type checking is that type-mismatch errors are generally not detected until runtime and then only if the program is thoroughly tested.

T F 5. Scheme's syntax eliminates the need for precedence rules.

T F 6. Perl uses ordinary infix arithmetic operators.

T F 7. One advantage of tail-recursion is the option of converting that recursion to iteration.

T F 8. In Scheme, the usual read-eval-print loop is applied to special forms as well as to functions.

T F 9. A function in Perl must have at least one formal parameter.

T F 10. Since it uses static scope, Scheme functions look up the values of non-local variables in the environment of their caller.

T F 11. Perl uses call by reference rather than call by value.

T F 12. In Perl expressions, any value that is not interpreted as False is interpreted as True.

T F 13. In Perl, the address of operator is used to access values stored in hash tables.

T F 14. Every Scheme procedure or function returns a value of some type, specified at compile time.

T F 15. In a Perl hash table, all of the values of keys must be unique.

T F 16. The values stored in a Perl hash table must be scalars.

T F 17. In Perl, pragmas may be used to give certain instructions to the compiler.

T F 18. In Perl, strings delimited by single quotes are syntactically identical to strings delimited by double quotes.

T F 19. The term "display" refers to the whole set of variables visible at a given point in a Lisp program.

T F 20. Because Perl arrays are immutable, their elements must be of the same type.

T F 21. Scheme's *delay* special form and *force* function can be used to implement streams.

T F 22. In LISP, the chomp operator is used to remove special characters from the input stream.

T F 23. In Java, call by reference is not possible since no pointer arithmetic is allowed.

T F 24. In Java, access to objects of a serializable class is controlled by semaphores.

T F 25. The C++ language allows for explicit definition of constructor and destructor functions.

## 2. Grammars (30 points: 10/10/10)

In the grammars shown below, upper case letters denote non-terminal symbols, lower case letters denote terminal symbols, the start symbol is always S, and ε denotes the empty string. For each grammar: (1) Describe the language generated by the grammar concisely in English. For example: "All strings of zeros and ones which contain at least 5 zeros." And (2) say whether or not the grammar is ambiguous. If the grammar is ambiguous, find a string that demonstrates the ambiguity, and draw two distinct parse trees for that string.

(a) $S \rightarrow aSa \mid bSb \mid cSc \mid a \mid b \mid c \mid \varepsilon$

(b) $S \rightarrow \varepsilon \mid ab \mid ba \mid aSb \mid bSa$

(c) $S \rightarrow aSb \mid C \qquad C \rightarrow \varepsilon \mid c \mid Cc$

## 3. Fill in the Blanks (20/ 10 * 2 points each)

1. When parameters are passed using call by _____, there is the possibility that a bug in the called function will cause the value of a constant in the calling function to be changed.

2. According to the principle of _____ hiding, an abstract data type should hide the details about how the data is represented internally.

3. In Java, the try statement is sometimes used with the _____ clause in order for certain actions to be taken whether an exception occurs or not.

4. In Java, which uses call by _____, copies of function parameters are made as functions are called.

5. Recursive descent compilation is an example of what is known as _____ parsing.

6. The class construct is C++ and Java allows data of different types to be aggregated, forming new types, known generically as _____ types.

7. In the definition of an _____ data type, there is an explicit connection between objects and operations defined on those objects.

8. In _____ arrays, the subscripts may be integers but other types may be used as well.

9. Some languages, such as Lua, allow functions to pass control to each other without necessarily getting it back. Such functions are known as _____.

10. Alan Turing's work on early computers played an important role in the Battle of the Atlantic during World War _____.

# 4. Writing a Lisp function (15: 10/5)

A classic LISP data structure is the Association List or ALIST. An ALIST is a list of items, each of which has a symbol and an arbitrary s-expression. For example:

((AGE 20) (MAJOR "CMSC")(COURSES (CMSC331 CMSC311 BIOL340))

(3a) Write a function ASSOC that takes two arguments, a symbol and an ALIST, and returns the item associated with that symbol (if any) in the ALIST or NIL if there is no item associated with the symbol. For example:

```
> (SETF A '((AGE 20) (MAJOR "CMSC")(COURSES (CMSC331 CMSC311 BIOL340)))
((AGE 20) (MAJOR "CMSC")(COURSES (CMSC331 CMSC311 BIOL340)))
> (ASSOC 'AGE A)
(A 20)
> (ASSOC 'COURSES A)
(COURSES (CMSC331 CMSC311 BIOL340))
> (ASSOC 'HATSIZE A)
NIL
```

(3b) Why is it a good idea to have this function return the pair (symbol + value) rather then just the second element of the pair (i.e., value)?

## 4. Functional programming I (25: 15/10)

```
> (MAXINT NIL)
0
> (MAXINT '(3))
3
> (MAXINT '(3 2 4 9 1 2))
9
```

Consider a function MAXINT that takes a list of positive integers and returns 0 if the list is empty and the largest value in the list if it is not empty. Examples of its behavior are shown in the box to the right.

**(4a)** Write a recursive version of this function in LISP without using any additional local variables. Hint: define MAXINT to take a list of integers, but have it call a recursive function MAXINT1 which takes two arguments – a list of integers and the largest integer seen so far. Three things to consider here: (1) what initial call to MAXINT1 should MAXINT make; (2) What should MAXINT1 return if the list is empty; (3) what should MAXINT1 return if the list is not empty? This will get you started:

```
(DEFUN MAXINT (L) (MAXINT1 …..))
(DEFUN MAXINT1 (…) …)
```

**(4b)**. The following is a recursive version of the maxint function in Perl. Fill in the blanks.

```
#!/usr/bin/perl
print "Recursive version of maxint function!\n";  # a short Perl example

use strict;  # an example of a pragma
use warnings;

_____  _________ {
   if (_______ == 0) {      # check for empty list
        0;
    } else {
        _____ $first = __________;   # save first element of input list
        my @newList = @_;           # make a copy of the input list
        _________ @newList;         # remove its first element
        my $max1 = maxint(@newList);    # recursion
        if (_____________ > ___________) {
            _____________
        } else {
            _____________
        }
    }
}

my @aList = (7,3,4,5);
my $max1 = maxint(@aList);
print "maxint of @aList is $max1\n";
```

## 5. Perl regular expressions (35 pts, 5/5/5/20)

Write regular expression patterns to match US social security numbers. Your patterns should match an entire string, leaving no characters unmatched. Please give your answers as assignments to variables P1, P2 and P3, e.g.,
P1 = '…your reg exp pattern goes here…'.

(a) Write a simple Perl regular expression, P1, that matches a SSN like "123-45-6789", i.e., three digits, a dash, two digits, a dash, four digits.

(b) Write another regular expression, P2, that matches SSNs where the separators between the numeric groups are optional and can be a mixed sequence of dashes or white space characters.

**Perl RegEx symbols**

| | |
|---|---|
| ^ | beginning of the string |
| $ | end of the string |
| + | one or more times |
| ? | optional |
| * | zero or more times |
| (...) | a group (with storing e.g. $1) |
| \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 digit |
| \w | an alphanumeric or underscore |
| \| | or |

(c) Write a final version, P3, that extends P2 and defines three capturing match groups, one for each of the three "fields" in the SSN.

(5d) Fill in the following table. For each row, assume that the following segment of Perl code has been executed:

```
($matching) = $string =~ /$pattern/;
```

The variable *$matching* will be matched to the result of the match, i.e., either undefined if the match failed or defined and given a value if it succeeded. Fill in the missing values in the table. Enter N/A in a cell if no answer is appropriate for it or it generates an error.

| $pattern | $string | defined($matching) | $matching |
|---|---|---|---|
| *a+(p)** | *apple* | *True* | *pp* |
| (\d*). | 12345 | | |
| (123\|abc) | 123abc | | |
| \w+\s(\w+) | foo bar x | | |
| (.*)y$ | xyxyx | | |
| a.([bc]+) | abcbcbcbc | | |

# LISP Cheat Sheet – condensed version

```
(defun helloWorld ()
  ;; prints the standard message
  (print "Hello World"))

(defun predicateDemo ()
  (print (member 3 (list 1 2 3 4 5 6)))  ; prints (3 4 5 6)
  (print (symbolp 'foo)) ; should be TRUE
  (print (eq 4 (+ 2 2)))  ; good for atoms
  (print (equal '(1 2 3) '(1 2 3)))  ; good for other things
  (print (null ()))  ; should be TRUE
  (print (listp '(1 2 3)))  ; should be TRUE
  (print (listp '3))  ; should be FALSE
)

;; basic lisp manipulation
(defun listDemo()
  (print (cons 'a 'b)) ; creates a dotted pair
  (car '(this is a list))
  (cdr '(this is another list))
  (cons 'quick '(brown fox))
  (append '(1 2) '(3 4))
  (print (reverse '(1 2 3)))
)

;; every language needs an if statement
(print (if (= (+ 1 2) 3) 'yup 'nope))

;; the cond statement does more!
(defun condDemo()
  (let ((a 2))
    (cond ((eq a 1) (princ "a is 1"))
          ((eq a 3) (princ "a is 3"))
          (t        (princ "a is something else"))))

;; example of mapcar
(mapcar #'sqrt '(1 2 3 4 5))
(mapcar (function sqrt) '(1 2 3 4 5))
; both return (1 1.4142135 1.7320508 2 2.236068)

;; example of apply
(apply #'append '((10 20) (30) (40 50)))
; returns (10 20 30 40 50)
```

# Perl Cheat Sheet – condensed version

```
#!/usr/bin/perl
print "Hello, world!\n";  # a short Perl example
# Perl statements end with a semicolon
# ordinary variables begin with $
my $nDocs = 0;
# perl is great for working with strings
# strings are delimited with either single quotes or double quotes
my $bString = "Inside double quotes, usual escapes apply";
# period . is the string concatenation operator
my $aLongString = $aString."\nconcatenated with the period operator\n".
"and the x factor for repetition\n$bString\nand interpolation!";

# LOTS of pattern matching operators, including s for substitute
$aString = "Do not do that!\n";
print $aString;
$aString =~ s/Do not/Don't/;
print $aString;

# nothing special about Boolean variables
# if it's zero, it's false, otherwise it's true
my $aBoolean = $fred lt $barney;

# control structures include if
# although we just defined $aBoolean, we can always test
if (defined($aBoolean) && $aBoolean) {
    print "fred is less than barney\n";
} else {
    print "fred is NOT less than barney\n";
}

# Perl supports lists and arrays, closely related concepts
# subscripts start at zero normally, as in C
my @fred;
$fred[0] = 2.8;
$fred[1] = "Wilma!";
$fred[2] = 'dino' x 2;

# push and pop add or delete elements from the end of a list
# use $fs to access an individual element of the list
# but we can use @fs to refer to the whole list
#
# sometimes we consider what Perl expects, i.e. list vs. scalar context
#
my @fs = qw/fred wilma barney betty/;
push @fs, qw/bambam pebbles/;
# shift and unshift delete or add elements to the start of a list
unshift @fs, "dino";
# list elements are separated by blanks when interpolated
print "@fs\n";
my @sf = reverse(@fs);
printf "Print the list in reverse @sf\n";

# simple echo of a given input file (default to STDIN) to STDOUT
sub echoDemo {
    # note use of @_ in both the scalar and list contexts in this if
    my $XLSfile = "default.xls";
    if (@_ == 1) {
      # list of arguments in list context
```

```
      ($XLSfile) = @_;
    } else {
      print "usage: echoDemo(inputFile=STDIN[,XLSfile=default.xls]\n";
    }

    my $word;
    my $nTerms=0;

    # Perl has built-in hash functions
    my %aHashTable = ();  # will be used in example below

    while (my $inputLine = <STDIN>) {
      chomp $inputLine;  # get rid of trailing newline
      print STDOUT "$inputLine\n";   #  I/O looks very C-like, eh?

      # recall that array names begin with @
      # split $inputLine into an array of blank-separated words

      my @words=split(" ", $inputLine);
      foreach $word (@words) {
          $aHashTable{$word} += 1;

          # to practice with regular expression matching, see if word
          # would be a good password, i.e. having at least one
          # digit, one lower case letter, and one upper case letter

          # make sure it finds a good password when run on itself XYzzy18
          if ($word =~ /[A-Z]/ && $word =~ /[a-z]/ && $word =~ /[0-9]/) {
            print "$word would be a good password.\n";
          }
      }
    }
    for $word (sort(keys(%aHashTable))) {
      $nTerms++;
      # but let's make each word lower-case
      # and make a variable wordlc local to this block
      my $wordlc = $word;
      $wordlc =~ tr/A-Z/a-z/;  # tr stands for translate
      printf STDOUT "%s, %d\n", $wordlc, $aHashTable{$word};
    }
}

#
# working with files and directories, and patterns
my @files = <*>;
foreach my $myFile (@files) {
    if (-f $myFile and -r $myFile) {   # see if it's a readable file
      my ($nl, $nw, $nch) =
          `wc $myFile` =~ /([0-9]+)\s+([0-9]+)\s+([0-9]+)/;
      print "file $myFile has $nl lines, $nw words and $nch characters\n";
    }
}
```