Review for Exam #1
CSCI-400 Spring 2013

The first exam will cover concepts of functional programming languages in general and by way of snippets of code written in Racket. Also covered will be concepts related to lexical and syntactic analysis including regular and context-free grammars, finite automata in general and lexers and parsers in particular. In addition, the exam will cover concepts related to the C programming language, though the emphasis is NOT on C syntax.

At least 25% of the points on the exam will come from questions on this review sheet verbatim. Note that included in this review sheet by reference are all homework assignments due on or before the date of the exam, so verbatim exam questions can be pulled from them and counted toward the 25%. Other questions are likely to be closely related. Some of the questions on the review sheet are not trivial and may take some time to figure out how to approach them at first. But if the first time you approach them is during the exam, that will be a choice you have made.

1) Write two functions, one called **`oddElements`** and the other called **`evenElements`**. Each takes a list (which you may assume is a (potentially empty) simple list of numbers). The first function returns a list consisting of the odd-numbered elements in the list while the second returns a list containing the even numbered elements. Note that odd/even here refer to the element's position within the list, not the value of the element. Hence:

```
>(oddElements `(1 6 8 3 10))
(1 8 10)
>(evenElements `(1 6 8 3 10))
(6 3)
```

Each function should return an empty list if passed a list too short to extract any elements from. You may find it helpful to make these functions mutually recursive (i.e., each calls the other), but this is not required.

2. What would be displayed by the following:

```
(cddar '((Alicia Brenda Charles)(100 Main Street)(Golden CO 80401)))
```

3. What is a higher order function and how is it different from a first-order function?

4. Write a recursive function named **mergeLists** that takes two ordered lists as parameters and returns a merged ordered list. Use a **cond** statement. You may assume that both lists are simple lists containing only numbers (though either or both might be empty). Example:

```
> (mergelists '(1 5 9) '(2 3 8 11))
(1 2 3 5 8 9 11)
> (mergelists '(1 5 9) '(1 3))
(1 1 3 5 9)
```

Review for Exam #1
CSCI-400 Spring 2013

5. Consider the following function:

```
(define (cl x)
  (let* [ (y 5) (z 20) (fn (lambda (y) (* x y z))) ]
        (let [ (y 10) (z (+ x y z)) ]
             (fn z)
        )
  )
)
```

a. What would be displayed by the call: **(cl 5)**

b. Explain your answer (i.e., explain how the values of y and z are set/changed as this function executes, explain what is displayed and what is returned. Be complete and specific). Explain how this is a closure.

6) Consider the following two functions:

```
(define (bob lst result)
  (if (null? lst)
      result
      (bob (cdr lst) (append result (list (+ 2 (car lst)))))
  )
)


(define (sue lst)
  (if (null? lst)
      '()
      (cons (* 2 (car lst)) (sue (cdr lst)))
  )
)
```

a. What would be returned by the call: (bob '(4 3 1) '())

b. What would be returned by the call: (sue '(50 25 10)

c. Which of the functions, if either, is tail recursive?

d. Why are tail-recursive functions valuable?

Review for Exam #1
CSCI-400 Spring 2013

7. Assume you have the following function:

```
(define (chooser op threshold)
  (lambda (x) ((eval op) x threshold)))
```

a. What would be displayed by the call: **(map (chooser '< 40) '(2 50 34 60))**

b. What would be displayed by the call: **(filter (chooser '> 20) '(1 2 30 40))**

c. Explain why and how this is a curry.

8. When is meant by "referential transparency"?

9. In Racket, parentheses, square brackets, and curly brackets (a.k.a., braces) may be used interchangeably to show grouping provided they are properly paired. Write the BNF rules for a grammar that enforces these rules. Assume that the alphabet (coming out of the lexer) consists only of these three pairs of grouping symbols and a single token called 'id'. In other words, 'id' can be a function name, an operator, a variable name, a literal value, or anything else that can legally appear within a set of parentheses.

VALID	INVALID
`[]`	`id`
`(id id)`	`(id id]`
`{id [] (id) {(id id)}}`	
`(id id [id] {[id [id id]] id})`	

10. Clearly define or describe the following concepts in such a way as to make it clear that you understand the similarities and/or distinctions: context free language, regular language, context free grammar, regular expression, finite automaton, production rule, terminal symbol, non-terminal symbol, token, and lexeme.

Review for Exam #1
CSCI-400 Spring 2013

For questions 11-20, choose the best option from the list below and enter the corresponding letter designation on the line provided.

A. Lexeme
B. Token
C. Symbol
D. Alphabet
E. Sentence
F. Parser
G. Lexer
H. Pushdown Automaton
I. Finite Automaton
J. Context Free Grammar
K. Regular Grammar
L. Production Rule
M. Regular Expression

11. _____ The set of all ASCII codes that could be present in a program's source code file.

12. _____ The machine capable of recognizing any context-free grammar.

13. _____ A processing engine that reads a source code file and produces a string of tokens.

14. _____ A terminal in a grammar.

15. _____ One or more symbols from the source code alphabet that, together, have a specific meaning.

16. _____ Defines the options that may be used to replace each non-terminal in a grammar.

17. _____ In general, a "program" constitutes a single one of these.

18. _____ A category of grammar that can be recognized by a finite automaton.

19. _____ A program that examines a string of tokens to determine a sentence's structure.

20. _____ The type of grammar used by most lexers.

Review for Exam #1
CSCI-400 Spring 2013

Grammar #1 (<rel_expr> is the start symbol)

```
<rel_expr> ➔  <expr> (lt|gt|eq|neq|lte|gte) <expr>
           ➔  <expr>
   <expr>  ➔  <term>
           ➔  <term> (add|sub) <term>
   <term>  ➔  <factor>
           ➔  <factor> (mult|div|mod) <factor>
 <factor>  ➔  id | open_paren <expr> close_paren
```

21. Using Grammar #1, what are the First() sets for this grammar? Is the grammar, as a whole, pairwise disjoint?

22. Using Grammar #1, draw the parse tree for the following expression

```
id add id mod id gt open_paren id sub id close_paren mult id
```

23. What is wrong with the following snippet of C code?

```
char *string;
string = "Fred";
string[2] = 'a';
```

24. What is wrong with the following snippet of C code?

```
char *string;
string = (char *) malloc( strlen("Fred")*sizeof(char));
```

25. What is wrong with the following snippet of C code?

```
char *myfunction(void)
{
      char string[12];
      strcpy(string, "Fred");
      return string;
}
```

26. Why is it important to perform a NULL pointer check after opening a file or allocating memory?

27. Describe, in detail, how to access command line parameters in C.

Review for Exam #1
CSCI-400 Spring 2013

28. At least one of these two grammars is ambiguous. Prove this by showing two different left-most derivations for the following sentence for one of the ambiguous grammars.

id * id / id

Grammar #1:

```
<expr>  → <expr> * <expr>     #1
        → <expr> / <expr>     #2
        → id                  #3
```

Grammar #2:

```
<expr>  → <expr> * id         #1
        → <expr> / id         #2
        → id                  #3
```

Grammer: #_____ is ambiguous

Derivation #1		Derivation #2	
Sentence	**Rule #**	**Sentence**	**Rule #**
<expr>	N/A	<expr>	N/A

Review for Exam #1
CSCI-400 Spring 2013

29. In the following grammar, uppercase characters are non-terminals and lowercase are terminals.

```
      #1     #2      #3
S  → aAC  |  BbbA  |  CAb
A  → Ccb  |  eB    |  BB
B  → cS   |  eA    |  fCbC
C  → bC   |  dBC   |  ggA
```

Non-terminal ->	S	A	B	C
First(#1)				
First(#2)				
First(#3)				
Pairwise Disjoint?	Y / N	Y / N	Y / N	Y / N

a. In the table above, list the First Sets for each rule and indicate, for each non-terminal, whether the rules are pairwise disjoint by circling the appropriate letter.

b. Is the grammar, as a whole, pairwise disjoint? Y / N

Review for Exam #1
CSCI-400 Spring 2013

30. What will be the output of the following Racket program?

```
(define (fnlst c)
  (let ((a 10) (b 20))
      (list (lambda (x) (set! a (+ c b)) (+ a b c x))
            (lambda (x) (set! b (+ c a)) (* 2 (+ a b c x)))
      )
  )
)

(define (work x)
  (let ((fns (fnlst x)))
    (let ((jack (car fns)) (jill (cadr fns)))
      (begin
        (display (jack 2)) (display "\n")
        (display (jill 2)) (display "\n")
        (display (jack 2)) (display "\n")
        (display (jill 2)) (display "\n")
      )
    )
  )
)

(work 1)
```