Midterm Exam – 100 pts **NAME:** KEY

CSCI-400 Spring 2013

SCORE:______ /100

For questions 1-10 (2 pts ea), 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

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

2. H The machine capable of recognizing any context-free grammar.

3. G A processing engine that reads a source code file and produces a string of tokens.

4. B A terminal in a grammar. Choice 'A' was also accepted

5. A One or more symbols from the source code alphabet that, together, have a specific meaning. Choice 'B' was also accepted

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

7. E In general, a "program" constitutes a single one of these.

8. K A category of grammar that can be recognized by a finite automaton.

9. F A program that examines a string of tokens to determine a sentence's structure.

10. K The type of grammar used by most lexers.

11) (15pts) 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:

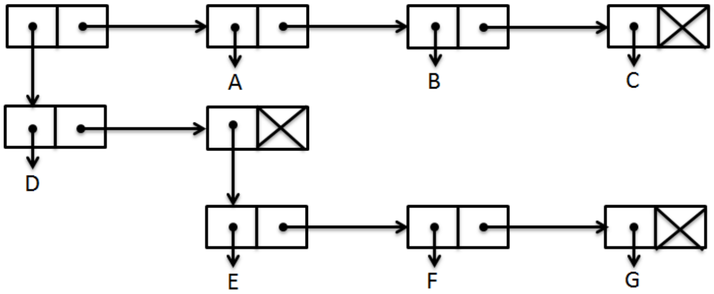
```
>(oddElements `(1 6 8 3 10))
(1 8 10)
>(evenElements `(1 6 8 3 10))
(6 3)
```

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.

```
(define (oddElements lst)
   (if (null? lst)
      lst
      (cons (car lst) (evenElements (cdr lst)))
   )
)

(define (evenElements lst)
   (if (null? lst)
      lst
      (oddElements (cdr lst))
   )
)
```

12) (10pts) Write a Racket literal definition (e.g., [define fred '(A (B C) D)]) that would produce the following data structure.



```
(define fred '( (D (E F G)) A B C) )
```

13. (5pts) 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))**

```
(#t #f #t #f)
```

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

```
(30 40)
```

c. Explain why and how this is a curry.

WHY: The 'chooser' function is a curry because it produces a function that performs the same task but with fewer arguments.

HOW: The 'chooser' function takes two arguments and embeds one of them into the single-parameter function that it returns.

14. (5pts) When is meant by "referential transparency"?

1) Any function/expression will always return the same result if given the same arguments.

2) Any function/expression can be replaced by any other function/expression that produces the same result.

Essentially, no external variables and no side effects.

Midterm Exam – 100 pts
CSCI-400 Spring 2013

Grammar (<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
```

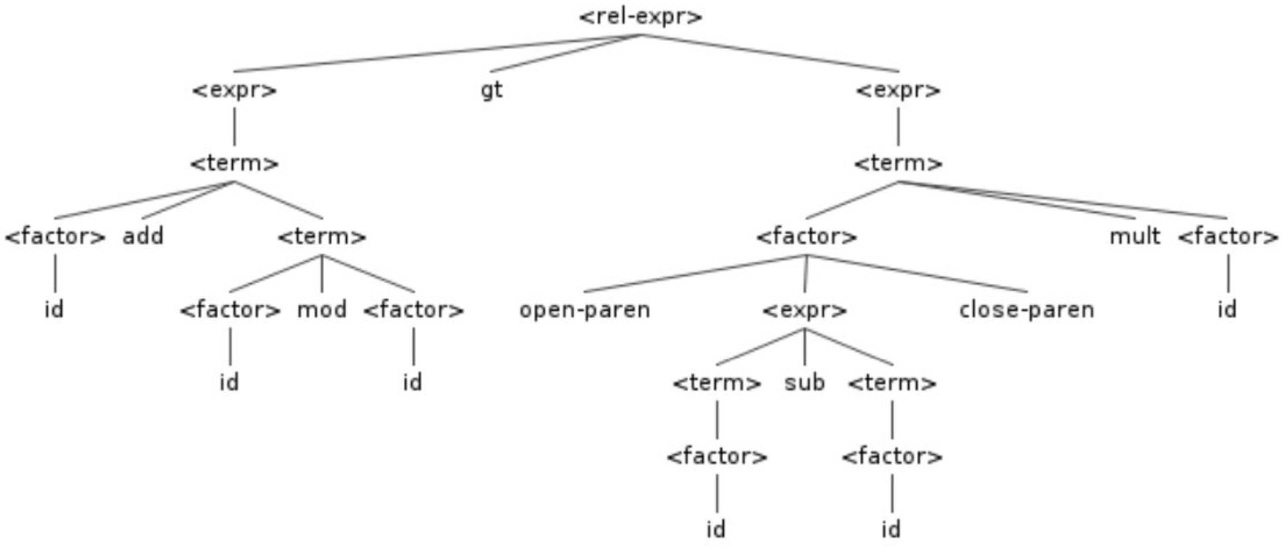
15) (16pts) What are the First() sets for this grammar?

```
<rel_expr> : { (id, open_paren), (id, open_paren) }
<expr>     : { (id, open_paren), (id, open_paren) }
<term>     : { (id, open_paren), (id, open_paren) }
<factor>   : { (id), (open_paren) }
```

16) (4 pts) Is the grammar, as a whole, pairwise disjoint?

NO, While <factor> is pairwise disjoint, the other three are not.

17) (15pts). Draw the parse tree for the following expression



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

18) (2pts) What is wrong with the following snippet of C code?

```
char *string;
string = “Fred”;
string[2] = ‘a’;
```

The last line is modifying a string literal.

19) (3pts) What is wrong with the following snippet of C code?

```
char *string;
string = (char *) malloc( strlen(“Fred”)*sizeof(char));
strcpy(string, “Fred”);
```

The malloc() call does not allocate memory for the NUL terminator.

20) (5pts) What is wrong with the following snippet of C code?

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

The function returns a local variable that is deallocated upon return.