Final Exam
CS400 Spring 2013 **NAME: KEY**

General Instructions:

Write clearly and legibly. If your handwriting (i.e., cursive) isn't clear and legible, then print!

Partial credit will be given when and where work is shown that makes the line of reasoning evident. If no work is shown, then no partial credit will be awarded. Note that work shown on one problem will not be considered for determining partial credit on another problem.

Unless otherwise indicated, the following apply to grammar rules:

1. Non-terminals are surrounded by angle brackets
2. Terminals are not surrounded by angle brackets.
3. Both BNF and Extended BNF forms may be used.

GRADING USE ONLY

PART 1: _____/ 100

PART 2: _____/ 25

PART 3: _____/ 50

PART 4: _____/ 25

TOTAL: _____/ 200

PART I – Multiple Choice

Choose the **best** option from the list below and enter the corresponding letter designation in the space provided **on this page**. Each answer may be used zero, one, or more times. Keep in mind that the question may or may not represent a defining relationship to the answer; in other words, the relationship may not be complete but, rather, more of an 'example' type relationship.

A. Closure
B. Lexeme
C. Syntactical analysis
D. First class functions
E. Currying
F. Right-most derivation
G. Token
H. Sentence
I. Parser
J. Lexer
K. Finite automaton
L. Regular grammar
M. Context-free grammar
N. Pushdown automaton
O. Production rule
P. Higher order function
Q. Lexical scope
R. Regular expression
S. Pairwise-disjoint
T. Alphabet
U. Lexical analysis
V. Referential transparency
W. Semantics
X. Grammatical ambiguity
Y. Left-most derivation
Z. Dynamic scope

1) T	6) V	11) Q	16) U	21) I
2) Z	7) J	12) O	17) C	22) K
3) M	8) R K	13) P	18) D	23) B G
4) A	9) G B	14) S	19) F	24) E
5) W	10) L	15) Y	20) N	25) X

(Answers in corner worth 2pts)

PART 1 (cont'd)

1. The set of all ASCII codes that could be present in a program's source code file.
2. When the referencing environment for a variable depends on the stack of functions that called the function containing the variable.
3. A parser generally works with this type of grammar.
4. When a function captures the references to variables that are within its referencing environment at the time the function is created.
5. Describes the meaning of sentence fragments.
6. When any expression may be replaced by any other expression that produces the same value.
7. A processing engine that reads a source code file and produces a string of tokens.
8. Used to describe/define a regular language.
9. A terminal in a grammar.
10. The type of grammar recognized by most lexers.
11. When the referencing environment for variables is determined by the structure of the source code only.
12. Defines the options that may be used to replace each non-terminal in a grammar.
13. A function that can accept functions as arguments, return values, or both.
14. When no two production rules for the same non-terminal can produce the same initial terminal.
15. When a sentence is produced by always expanding the first non-terminal in the sentence.
16. The process of examining the source code to determine if it represents valid tokens in a grammar.
17. The process of examining the tokens in a sentence and deciding if they form a valid sentence.
18. A language is said to have these if functions can be treated like any other value, including being passed as arguments, returned by functions, and assigned to variables.
19. Bottom-up parsers work with this type of derivation.
20. The machine capable of recognizing any context-free grammar.
21. A program that examines a string of tokens to determine a sentence's structure.
22. A machine that can recognize any regular grammar.
23. One or more symbols from the source code alphabet that, together, have a specific meaning.
24. The process of producing a function that wraps another function is such a way that the number of arguments needed is reduced.
25. When two different derivations of the same type produce the same sentence.

PART 2

Consider a unambiguous context-free grammar that contains the three tokens '**#**', '**$**', and '**&**'. The '**#**' token might be a literal or a variable name and can be used as an operand, while the '**$**' and '**&**' are binary infix operators that can produce values that can be used as operands. The '**&**' operator has precedence over '**$**'. The '**$**' operator is right associative while the '**&**' operator is left associative. There are two non-terminals in the grammar, **<start>** and **<other>.** The **<start>** non-terminal is the Start Symbol. Recall that "infix" merely means that the operator appears between the two operands.

2.1 (10pts) Which of the following productions are consistent with this grammar?
Check all that apply – 1pt for each correctly checked/unchecked item.

X	<start> => <other> $ <start>	X	<other> => <other> & #
___	<start> => <start> $ <other>	___	<other> => # & <other>
___	<start> => <other> & <start>	___	<start> => #
___	<start> => <start> & <other>	___	<other> => <other> $ #
X	<start> => <other>	X	<other> => #

2.2 (15pts) Each of the expressions below presumably represents the natural grouping of a grammar similar to the one above, meaning that the grouping matches the order of evaluation of the expression **a & b $ c $ d & e & f $ g** in that grammar. For each expression determine which operator has higher precedence and also what the associativity is for each operator. Circle the appropriate symbol, choosing the question mark if the answer cannot be determined from the expression, either because the needed information is not there or because it is contradictory.

Expression	Order	&	$
(((a & b) $ c) $ ((d & e) & f)) $ g	(&)$?	(L)R ?	(L)R ?
a & ((b $ (c $ d)) & (e & (f $ g)))	&($)?	L(R)?	L(R)?
(a & b) $ (c $ (((d & e) & f) $ g))	(&)$?	(L)R ?	L(R)?
(a & (b $ (c $ d))) & (e & (f $ g))	&($)?	L R(?)	L(R)?
a & ((b $ c) $ ((d & e) & (f $ g)))	& $(?)	(L)R ?	(L)R ?

PART 3 - (5 pts each)

3.1) Why shouldn't the variable yytext be referenced in the Bison input file?

The variable yytext is a global variable that stores information about the most recent token that has been read from the file stream. The code in Bison needs to work with the token information that was previously pushed onto the stack. Thus, the lexer needs to pass parser dynamically allocated copies of the yytext content.

3.2) Why are stack-dynamic variables generally referred to as "automatic" variables.

Memory for stack-dynamic variables is automatically allocated on the function call stack as functions are called and deallocated when they return.

3.3) What is the difference between a formal parameter and an actual parameter?

Formal Parameter: A placeholder variable in a function definition that receives the value of the passed argument.

Actual Parameter: The value that is passed to a function when the function is called.

3.4) In C, the statement `printf("%f", x);` works the same way regardless of whether x is a float or a double. Why?

Because if 'x' is of type float, then it is implicitly coerced to type double when it is passed because it corresponds to an ellipsis in the function prototype.

3.5) In C, when an array is passed to a function and the formal parameter is not declared merely as a pointer to the type of data stored in the array, the declaration must include the number of elements in each dimension except one, which may be left blank. Which dimension is optional and why is it not necessary?

The first dimension is optional because it represents the number of rows in the array. This is not needed for two reasons:

(1) C arrays are row-major and hence this information is not needed to calculate the offset of an element from the start of the array

(2) C does not perform array bounds checking, so the total number of rows is not needed for that purpose.

3.6) What is printed by the following C statement:

```
printf("%s", &(6+5)["Hello World"]-4);
```

answer: orld

```
char *H = "Hello World";
printf("%s", &H[11]-4);
printf("%s", H+11-4);
printf("%s", &H[7]); // H[7] = 'o' in "World"
```

3.7) In C, how are arguments corresponding to an ellipsis as the formal parameter list handled?

```
float -> double
char, short -> int
```

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

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

The third line is modifying a string literal.

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

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

No space is allocated for the NUL terminator.

3.10). 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 pointer to an automatic local variable that will be deallocated when the function returns.

PART 4 (5 pts each)

4.1) What is meant by a "dangling pointer" and by a "memory leak"? What problems can each cause?

Dangling Pointer: A pointer that points to memory that has been deallocated. This can cause data corruption, the memory manager to fail, or access violations.

Memory Leak: All references to a memory block are deleted but the memory is not deallocated. Can cause performance problems and unnecessary out-of-memory errors.

4.2) What is meant if a language is said to exhibit "guaranteed short circuiting"?

When a Boolean expression is evaluated, the order of evaluation of the operands is specified (generally left then right) and the second operand is not evaluated if the value of the expression can be determined from the first operand alone.

4.3) What is a "widening" conversion and what is a "narrowing" conversion?

Widening: ALL values of the type being converted from can be approximated in the new type.

Narrowing: At least some value of the type being converted from do not have approximate representations in the new type.

4.4) What does a "referencing environment" refer to?

The collection of all variables that are visible to an expression when it is being evaluated.

4.5) What is the basic requirement that must be met in order for an expression to be used as an lvalue (and do not just say that it has to be able to appear on the left side of an assignment operator)?

The expression must yield a value that represents an address that can be the target of an assignment operation.