C M S I 4 8 8

Final Exam

The test is open-everything with the sole limitation that you neither solicit nor give help while the exam is in progress.

The final exam is in two parts.

- Part I: Do all 10 problems below. You may not take more than 3 hours on the exam from the time you first look at the problems. Submit all answers by 9:00 a.m. on Saturday, May 10, 2014. Each problem is worth 10 points.
- Part II: Meet at the gym at 9 a.m. on Thursday, May 8, 2014, for one hour of full-court basketball. Students that shoot more than 50% from behind the three point line during a game (4 shots minimum) receive a 10 point bonus. Students shooting 75% from behind the three point line during a game (again, 4 shots minimum) receive a 40 point bonus.

The basketball hour portion of the CMSI 488 final is an LMU Computer Science tradition dating back to 1998. Everyone plays and everyone has fun. It is a nice way to relieve stress during finals week, and is part of the education of the whole person — mind, body, and spirt — that is part of the LMU Mission Statement.

---

1. Sketch the AST for the following Java fragment:

```
class Y {
    public static void main(String[] args) {
        (new Y[]{new Y()})[7].main(new String[0]);
    }
}
```

2. Write regular expressions for
   a. Octal constants in C. (If you don't know what these are, be happy this is an open-book, open web test.)
   b. Unsigned binary numbers divisible by 16.
   c. 16-bit hexadecimal numerals divisible by 8 (signed or unsigned!).
   d. Floating-point constants that are not allowed to have an empty fractional part and can have no more than three digits in the exponent part.
   e. The set of all character strings that contain neither the substring `"file"` nor the substring `"final"`, case insensitve.

3. Consider the following little language of object literals. Objects are key-value pairs separated by commas and enclosed in curly braces. Keys are identifiers (non-empty strings of ASCII letters, digits, and underscores that must start with a letter), and values are either numeric literals, string literals, boolean literals, identifiers, other objects, or arrays (comma-separated values enclosed in square brackets). For simplicity, assume string literals are just character sequences enclosed in double quotes, with no special escape sequences; all characters except control characters are allowed. This little language has no comments. An example object description is:

```
{
  dog : "sparky",
  id : 3,
  pups : [24, 188, "spot", 3.2e5, false],
  house : {color : "GREEN", date : {m : 4, y : 2015, d : 22}},
  nothings : [{}, {}]
}
```

Write

a. A microsyntax
b. A macrosyntax
c. The recursive descent parsing functions that you need. Please assume that the `match` and `at` functions have already been written for you, as well as the *entire scanner*. In other words, you aren't being asked to write very much here. Again, write *only* the parsing functions.

4. In Java, we write array literals like so:

```
new int[]{4, 6, 8}
new boolean[]{true, false, isPrime(x)}
new Point[]{new Point(0,0), new Point(3,4), new Point(3,0)}
```

   Why did the designers of the language require the type? Hint: Think about what is required to do semantic analysis. Do not give a long-winded answer. You need only give a single sentence.

5. Classify each of the following conditions as a lexical error, a syntax error, a static semantic error, a dynamic semantic error, or just fine.
   a. The expression `[1,2,3][1]+[2]` in JavaScript.
   b. A string literal with a non-breaking space in it in Java.
   c. `x > y > z`, where `x` and `y` are `int`s, and `z` is a boolean, in Java.
   d. Calling a method on an integer literal in JavaScript (Note: Tricky!)
   e. The Java declaration `String `ohana = "family"`.
   f. A return statement at the "top level" of a script in Java.
   g. The statement `f(x)[3] = 1`, where f is a function of one integer argument returning an integer array in Java.
   h. Returning a function from a function in Python.
   i. The expression `b.x` where `x` is not a field of the object `b` in JavaScript.

6. Here's the macrosyntax of a little language:

```
SCRIPT  → FORM+
FORM    → '(' 'define' ID PARAMS FORM+ ')' | '(' OP OPERAND+ ')'
OP      → '+' | '-' | '*' | '/' | ID
OPERAND → NUMLIT | STRLIT | ID | '(' OP OPERAND+ ')'
PARAMS  → '(' ID* ')'
```

   Turn this basic grammar into an attribute grammar in which all FORM elements have an attribute called `level` storing the nesting depth of their enclosing function. Forms at the level of a script have a level of 0; forms inside a function declared at the top level have a level of 1, and so on.

7. Here's something we actually talked about in class, on a day when only about half of you showed up. I want to reward you for attendance, so here goes. Consider the problem of designing a language syntax to support compound statements like `if` and `while`.
   a. There are four main approaches. One is the "curly brace language" approach. What are the other three?
   b. Because code in curly brace languages can look like ~~shi~~crap when one sometimes puts in braces and sometimes does not, the designer of Perl made braces required for *all* such statements. Yet, he also created a new keyword, `elsif` for multiway conditionals. Why did he think he needed the `elsif`? What problem was he solving?
   c. Not that I think I am smarter than Larry Wall, but I've actually designed curly brace languages that do not use `elsif`; I just do this:

```
IFSTMT -> 'if' '(' EXP ')' BLOCK
          ('else' 'if' '(' EXP ')' BLOCK)*
          ('else' BLOCK)?
BLOCK -> '{' STMT* '}'
```

      Does that solve the problem Wall solved with `elsif`?
   d. How does my approach work in a terminating-end syntax language (like Ruby, Lua, Fortran, or Modula)? Hint: I would get this grammar:

```
IFSTMT -> 'if' EXP 'then' STMT+
          ('else' 'if'  EXP 'then' STMT+)*
          ('else' STMT+)?
          'end'
```

      Does it look okay? Is this grammar left recursive? Is it LL(k)? Why or why not?

8. Consider a language for describing vector graphics. An example program in this language (formatted ugly to highlight the fact that line breaks do not matter) is:

```
down deg color 1 0 0 left
90 forward 4 color 0 0 1 [ left 90
forward 1.5 ] right 90 forward 1.5 up
```

This program draws the letter T with a red vertical line of size 4 units and topped with a 3 unit blue line. A program is a sequence of instructions. The instructions are:

`deg`	switch to degree mode
`rad`	switch to radians mode
`down`	put the pen down so movements draw lines
`up`	pick the pen up so movements don't draw anything
`left θ`	turn counterclockwise by angle `θ`
`right θ`	turn clockwise by angle `θ`
`forward n`	draw a line by moving forward n units.
`backward n`	draw a line by moving backward n units.
`color r g b`	set color (r,g,b), values are floats in the range 0 to 1.
`[`	save current state
`]`	restore previously saved state

Write a macrosyntax in EBNF for this language. Handle the brackets reasonably, please. State whether your grammar is LL(1) or not, and whether it is ambiguous or not. Again, just the macrosyntax.

9. Consider the addition of *parallel assignment* to Iki. That's right, we need to be able to write the awesomeness that is `x,y = y,x`. To support this feature:
   a. Show changes to the macrosyntax.
   b. Do you need a new entity module? If so, give it. If not, which existing entity class do you add to?
   c. Show changes to the parser. Just the parser. Not the scanner. Just the parser.
   d. What static semantic rules must be written into the specification? (Don't implement the `analyze` methods, just tell me what you would have to do during analysis. Well, okay, if you can best tell me by writing code, do so.)
   e. Give the JavaScript translation for the new Iki expression `a, b = b, a+b`.

10. Given the following script, written in some generic imperative language (okay yeah it's Python):

```
x = input("Enter a number")
y = x * 3 - 7 + x * 3 - 7
b = ['c']
for x in "rats" * (y * 2):
    b.append(x)
    b.append(x)
print b
```

write the most amazing JavaScript translation you can for this script. You will have to bring in some of your own intelligence to bear on this problem. You may assume the existence of a function called `readInt` in JavaScript. Also, note that the `*` operator is overloaded in Python. You might need to look up the meaning of "multiplying" a string by an integer.