C M S I 4 8 8

Midterm Answers

1. All you need to do is copy over the existing EBNF and change the right hand sides, simplifying a little because we don't read or write lists, only individuals. Also, there's no indication of how calls look, so we'll skip those. And keep in mind: the question DID NOT ask for microsyntax, only macrosyntax.

```
PROGRAM   ->  BLOCK
BLOCK     ->  (DEC '.')* (STMT '.')+
DEC       ->  'DECLARE' 'INT' ID
STMT      ->  'MOVE' EXP 'INTO' ID
          |   'READ' 'FROM' 'STDIN' 'INTO' ID
          |   'WRITE' EXP 'TO' 'STDOUT'
          |   'WHILE' EXP 'IS' 'NOT' 'ZERO' 'LOOP' ':' BLOCK 'END' 'LOOP'
EXP       ->  TERM (('PLUS' | 'MINUS') TERM)*
          |   ('SUM' | 'DIFFERENCE') 'OF' TERM 'AND' TERM
TERM      ->  FACTOR (('TIMES' | 'DIVIDED' 'BY') FACTOR)*
          |   ('PRODUCT' | 'QUOTIENT') 'OF' FACTOR 'AND' FACTOR
FACTOR    ->  INTLIT | ID | '(' EXP ')'
```

2. To make

```
x x = x.x;
```

work in Java, we would need a class x with a field x. The type of the field x would have to be the class x. This is easy:

```
class x {
    x x;
    ...
}
```

But the problem is now when we try to declare a new variable x, for example:

```
public void f() {
    x x = x.x;
}
```

we get

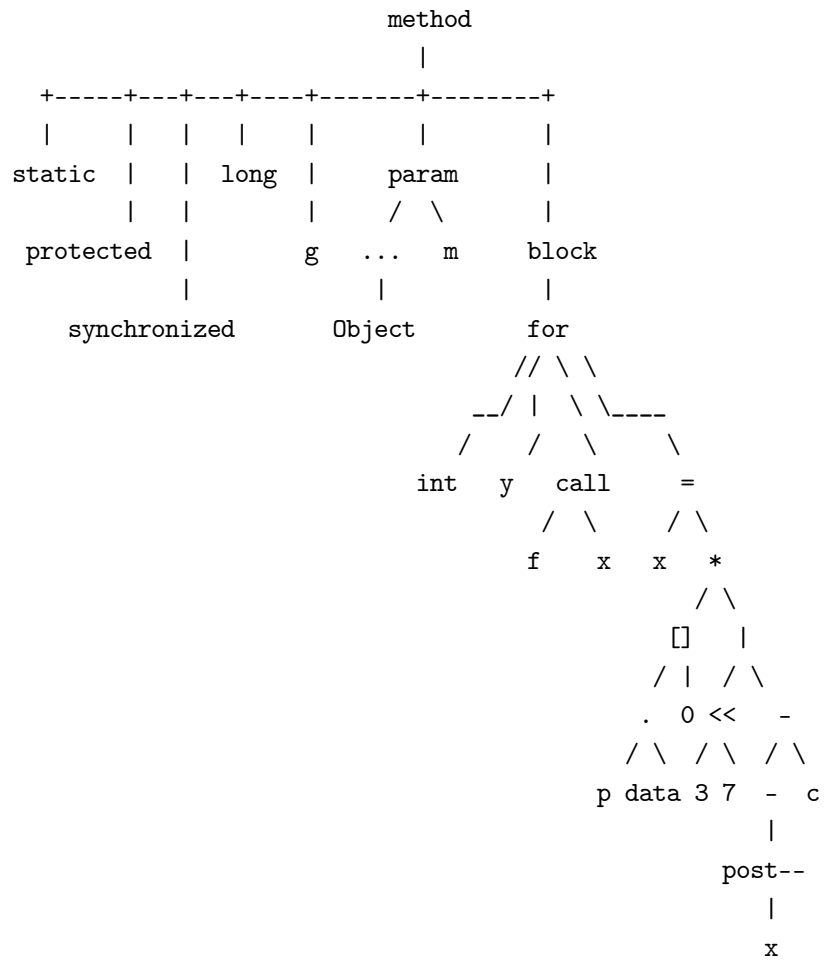
```
error: variable x might not have been initialized
```

because we're trying to read the x field of x within its own declaration! So we can't do this. But if x is declared first and then we try to assign:

```
x x = new x();
x = x.x;
```

we are good!

3. a. **syntax error**, because `super` is a reserved word.
   b. **static semantic error**, because parameters and local variables are in the same scope and this code fragment is redeclaring x within a score. (For fun, note this is not an error in C.)
   c. **lexical error**, because ` is not an allowable character.
   d. **no error**, as we're just storing a reference to a string object in the variable pet. We didn't change the inside of any string, so we're not violating immutability rules.
   e. Well two answers apply here. You can say **syntax error** because there is a missing semicolon or you can say **static semantic error** since strings are immutable.

4. 

5. You needed to understand that this problem was talking about array and object expressions in a statically-typed, strongly-typed, language. Now statically-typed means that the **types of all expressions are known at compile-time** (you knew this!) and so the compiler needs, during semantic analysis, to compute the type of every expression and store that type in the expression object.

   Now in the problem you saw array and object expressions that did not type names attached to them. So how the heck is the compiler supposed to know the types of array expressions? Well, sometimes it is easy. For example, the compiler can easily determine that the type of

   ```
   [4, 6, 8]
   ```

   is "array of integer" and the type of

   ```
   [5.67, -2,22, 9, 0,9]
   ```

   is "array of double" but what is the type of

   ```
   ["orange", 2, 3.3, [1,2,3]]
   ```

In order to assign a type like this, we have to **introduce the notion of computing the least general type that covers all array elements**. In this case we would then compute the type as "array of object" (where object is a type including as subtypes strings, ints, floats, other arrays, and so on. So to implement this feature, the compiler would compute the proper "least general" type for the array, and likewise for objects, would need to come up with a mechanism to store the **type signature** of an object containing typed key-value-pairs. We would need new type compatibility rules to deal with things like extra or missing fields. It can be done but it is tedious.

6. I actually meant to write

```
G -> (S s G)?
S -> V q E | r i | w E | u E S+
V -> i | V d i | V '[' E ']'
E -> n | V | '[' E* ']' | '{' (i E)* '}'
```

but I ended up writing

```
G -> (S s G)?
S -> V q E | r i | w E | u E S+
V -> i | V d i | V a E a
E -> n | V | a E* a | b (I E)* b
```

   a. Yes, `waiaaaaas` has multiple parse trees. However, I did not take points off for a wrong answer here because it was kind of hard to find this. The grammar I intended to give you was simpler and not ambiguous.

   b. No, it is not LL(k) because it is left-recursive.

   c. Here you have to change the grammar because JavaCC cannot handle-left recursion. The new rule is

```
V -> i (d i | a E a)*
```

   so the JavaCC is:

```
void V() {} {"i" ("d" "i" | "a" E() "a")*}
```

   d. It looks like S is statement, s is a semicolon, V is variable, q is the assignment operator, E is expression, r is read, i is identifier, w is write, u is until, d is dot, n is a number. So it is the macrosyntax of a little language made up of a sequence of statements (assignment, read, write, and until). Variables are either simple identifiers or of the form `arrayvar[exp]` or `objectvar.field`. Expressions are numeric literals, variables, array expressions, and object expressions.

7. a. `0[0-7]*`

   b. `\d+(\.\d*)?([Ee][-+]?\d{1,3})?`

   c. `^((?!exit|exec).)*$`

   or

   `(e(?!xit|xec)|[^e])*`

   d. `[01]{29}000`

   e. `([a-z]+)\1`