CSE305, Fall 2007 **Assignment 3** **Due Wed. Sep. 19**

Final Recitations and Office Hours

(R1) Mondays, 10–10:50am in the CSE Commons Conf. Rm. from 9/24 on, but 242 Bell Hall on 9/17.

(R3) Tuesdays, 10–10:50am in the CSE Commons Conf. Rm. from 9/25 on, but 242 Bell Hall on 9/18.

(R2) Tuesdays, 2–2:50pm in 260 Capen **(unchanged)**

Recitation (R4) is being kept on the books to prevent re-registration headaches. It does not matter which of (R1,R2,R3) you actually attend—there is no need to change your registration. The CSE Commons Conference Room is on the ground floor of the main brick UB Commons building, at the end of the spur in front that points north toward the Ellicott Complex. It has the wall nameplate "Truthing Lab." You can enter from the inside (go to the end of one long hall, then jag left-right and go to the end of the next hall) or from the outside door facing the bookstore parking lot—if both are locked, knock on the latter! **But the room will not be ready next week, so next week's recitations (R1,R3) are in Bell Hall again.** Recitation R2 meets in 260 Capen as-usual.

Final Office Hours

Regan: Mondays 1–2pm, Tuesdays 10am–noon, Wednesdays 1–3pm.

Lollett: Tuesdays noon–2pm.

Notice that office hours like recitations are stacked before most assignments will be due. (If we have a project due on a Friday afternoon, we may use the R4 Fri. 4–4:50pm time and 260 Capen place for absolute last-minute help, but my general policy is against that.) I may open extra hour(s) in some weeks, e.g. before the Fri. 10/12 prelim exam, and actually my Monday 1–2pm hour is still subject to change. CSE305 people have priority over CSE396 on Tuesdays; CSE396 people have priority on Wednesdays for homeworks due on Thursdays.

Reading:

Next Monday's lecture will still be in my "Compilation" notes which substitute for Section 3.4 and parts of Chapter 4 (which are still worth *skimming*), but Wednesday's lecture may turn the page into Chapter 5 and the *content* of programming languages proper, so please read Chapter 5 for next week. (You will find that when I hit Chapter 6 the week after (i.e., the last week of September), suddenly I'll be going much slower relative to the text.)

(1) Let $L_0 = \{ a^n b^n : n \geq 1 \}$ abbreviate the language for which you wrote a short BNF grammar (call it G_0) in problem (3) of Assignment 2, i.e. ex. 13 on p172 of the text. Now define $L_1 = \{ x\#y : x \in L_0 \text{ or } y \in L_0 \}$. Here '#' is a new terminal character, and standardly the 'or' is inclusive, not exclusive.

(a) Design a BNF grammar G_1 such that $L(G_1) = L_1$. (9 pts.)

(b) Is your grammar G_1 ambiguous? If so, find an ambiguous string and show two different parse trees for it in your G_1. If not, explain why your G_1 is unambiguous. (9 pts., for 18 total)

(2) This exercise refers to my condensed BNF for Java 2.0 (aka. 1.2), which was given out in class and is also on the "Language Links" part of the course webpage. It also refers to the grammar of the *C♯ Language Specification, ECMA-334 4th. ed.*. (It is the top Google hit for `"ECMA-334"`, and the most relevant pages are 278–281 and 451 of the document, which are pages 300–303 and 473 of the PDF file. **Please do not print this yourself**—sections of the C♯ grammar have been posted to the newsgroup, and some will be given in another handout on Friday. Also, you need not know what `strictfp` means—it refers to two levels of compliance with IEEE floating-point standards.)

(a) Which of the following are legally derivable from the indicated nonterminal symbol (i.e., programming language category)?

i. `COMPUNIT` $\Longrightarrow^{*?}$ `class Foo {strictfp float x;}`

ii. `COMPUNIT` $\Longrightarrow^{*?}$ `class Foo { strictfp class Bar {} }`

iii. `FIELD` $\Longrightarrow^{*?}$ `int x = 3, y = 3;`

iv. `FIELD` $\Longrightarrow^{*?}$ `int x,y = 3;`

v. `FIELD` $\Longrightarrow^{*?}$ `int myArray[] = {1,2,3};`

vi. `FIELD` $\Longrightarrow^{*?}$ `int[] myArray = {1,2,3};`

For each, give a yes/no answer and then justify it with either a derivation or a *brief* prose answer. For instance, the ridiculous `class Silly{ { {} ; ; {} } }` is derivable from `COMPUNIT` because a COMPUNIT can be a single class, a class can have the form "`class ID { CDEC }`" with a single `CDEC` inside, a CDEC can be a BLOCK, which introduces the second pair of nested `{ ... }`, and `;` and `{}` are legal `STMT`s! ($6 \times 3 = 18$ pts.)

(b) One of the last four items (c)–(f) cannot be derived in the grammar of C♯—and this is correctly reflected by the `mono` implementation on `fork/yeager`. Find it, and give a brief explanation of what the C♯ grammar is missing. (6 pts.)

(c) Going back to my Java 2.0 grammar, suppose the rule `CASTEXP ::= "(" TYPE ")" EXP4` were changed to `CASTEXP ::= "(" TYPE ")" EXP3`. What kind of lexical ambiguity could result? Hint: TYPE derives NAME, and an expression can also be a NAME. This explains why the nonterminals `EXP3` and `EXP4` have to be separated in the Java grammar—and this difference is also reflected in the official Java 2.0 grammar. (6 pts., = 30 on this problem.)

(3) In C, C++, and Java, the modulus operator `%` has equal precedence with `*` and `/` and is left associative, while the shift operator `>>` has lower precedence, i.e. "binds looser than," addition and subtraction. Extend the grammar below (which is basically part of Example 3.4) to create an unambiguous grammar for expressions with these two additional binary operators (besides `+,-,*,/` that is). Give a leftmost derivation of `r%2*n>>d-1` and explain how it is parsed. (18 pts.)

```
E  ::=  T  |  E+T  |  E-T
T  ::=  F  |  T*F  |  T/F
F  ::=  -F |  (E)  |  any-constant-or-variable.
```

(4) With reference to the ML BNF handout, also given out in class and linked from the language page (plus the relevant portion has been posted to the newsgroup), give both a leftmost derivation and a parse tree for the following ML code:

```
fun qsolve(a,b,c) = (~b + sqrt(b*b - 4*a*c))/2*a;
```

You need not know anything about ML besides the grammar, and while the above is legal in the grammar, there are actually 2 errors that care caught at a later stage of compilation than parsing. (12 pts., for 78 total on the set)