DUE: Monday 6 February 2006

READING

- Complete the reading assigment regarding syntax listed on the web page.
- Watt text: Chapter 3 (Semantics) — Sections 3.1-3.2; Chapter 5 (Theory) — Section 5.1
- Scott text: Chapter 3 (Names, Scopes and Bindings)

PROBLEMS

General Instructions for all written work in CSc 520: Please submit your solutions to homework assignments using a program capable of producing typeset output with the appropriate type faces, symbols and notations peculiar to programming language semantics (e.g., `TeX`, `LaTeX`, `troff`, `MSWord`). *Do not submit handwritten work.*

Diagrams or pictures can be done free-hand; a drawing tool like `xfig` is preferable.

Start each solution on a *new page*, identify the problem clearly, and *number each page*. Use *one side* of the paper only. Submit answers in the envelope that is labeled with your name (these envelopes will be provided in class.) *Do not seal* the envelope, so that it may be reused for several assignments.

Clarity and conciseness in writing will earn points, so revise your work before submission, and consider how your work is presented.

Some problems require syntactic or semantic specification. Your syntactic/semantic descriptions should include thorough comments to explain the ideas behind each specification in clear technical English. The existence of a formalism for describing syntax or semantics does not relieve you of this responsibility, any more than it does when you are writing code. Well-chosen examples are never a bad idea, when a difficult point is being explained.

In some problems, a range of different but equally reasonable semantics is possible. Grades will not be based on the "rightness" of any particular choice, (as long as it is sensible, to the point, and does not trivialize the problem) but on how clearly and thoroughly you describe your choice, both informally in your explanation and rigorously in your semantic/syntactic descriptions.

Some problems require ‘‘experiments’’ to find out what actually happens in a compiler, or to test the meaning of a construct as it is actually implemented. For such experiments, always give the full context: particularly the specific compiler used, version and date, manufacturer and the hardware and OS used. This is particularly important if you use compliers other than the ones installed on `lectura`.

1. Make

Scott text, Problem 1.5.

2. Machine Dependence

Scott text, Problem 1.4. Answer for Java and Scheme (not ADA and Pascal).

3. Expression and Assignment Semantics in Pascal, C++, C

In IEEE/ANSI Standard PASCAL†, an assignment statement has the form

$$variable-access := expression$$

A "variable-access" is one of four things: an "entire-variable", which is just a simple identifier like *x* or *foo*, a "component-variable" like $a[i]$ or $ptr\uparrow.data$, an "identified-variable" like $ptr\uparrow.father\uparrow$, or a "buffer-variable" like $input\uparrow$. The official dictum (§6.8.2.2) is: ‘‘The decision as to the order of accessing the variable and evaluating the expression shall be implementation-dependent: the access shall establish a reference to the variable during the remaining execution of the assignment-statement". You can’t change horses in mid-stream, but

†American National Standard Institute, *IEEE Standard Pascal Computer Programming Language: an American National Standard* (ANSI/IEEE 770 X3.97-1983), New York: Institute of Electrical and Electronics Engineers, 1983.

you get to say which bank the horse starts on.

(a) Could there be a situation in Pascal where this evaluation order makes a difference in the outcome of the assignment? Could this affect portability of code? If so, give a concrete example. If not, explain why not.

(b) Design an experiment to discover the "implementation-dependent" evaluation order for a compiler to which you have access, and explain your findings. Edit and comment your output listing for clarity and compactness. Do not include irrelevant commands, but do include adequate information (file contents, etc.) to allow the experiment to be duplicated.

The assignment statement‡ in C++ can be more complex, partly because there is a richer repertoire of legal expressions.

(c) What is the standard for C++ that is analogous to the discussion above of ANSI PASCAL? Is the evaluation order in an assignment statement determined by the language semantics? What is guaranteed by the semantics, and what is not?

(d) Give the meanings of the following two fragments, or discuss any difficulties involved, based on your answer to (c):

```
i = 7; i = v[i++];

i = 7, i++, i++;
```

(e) In the C language, the C Standard says that it is undefined to modify a variable or location, and then try to *use* its value at another point, *unless* it is definitively known which of these actions (modification or use) will occur first. So, for example, `i++ * i++` does not meet the standard.

Discuss the assignment `a[i] = i++;` in terms of this prohibition in the Standard. What are the possible meanings of this fragment? For each of these possible meanings, give C code that will *force* that outcome to occur on *any* C compiler.

(f) You have learned from above that some programming languages do not completely specify the evaluation order in arithmetic expressions and assignment statements. Based on your knowledge of instruction sets and how machine code is generated by a compiler, what is the reason for this ‘‘underspecification’’ in programming languages? Give a concrete example where such ‘‘underspecification’’ would be important (or equivalently, where complete specification would create difficulties.)

4. Semantics of Array Assignments

When ‘‘structured’’ variables, such as arrays or records, are involved, the semantics of even the assignment statement becomes complex. If you do not believe this, consider the following problem.

(a) Just after the following C/C++ assignment executes, is it *always* true that `(a[ a[i] ] == 1)`? That is, it is true in *all possible contexts*? If yes, explain in detail the effect of executing the above. If no, give a counter-example.

```
a[ a[i] ] = 1;
```

(b) Can the order of evaluation of sub-expressions affect the meaning of this assignment?

(c) One might conjecture that a reasonable semantic rule for assignment in C/C++ would state:

- the *r*-value of *targ* **after** executing assignment *targ* `=` *source*`;` is equal to the *r*-value of *source* **before** executing the command.

This sounds eminently reasonable. Is this conjectured rule correct? [*HINT:* Consider the assignment in part (a).]

‡Margaret A. Ellis and Bjarne Stroustrup, *The Annotated C++ Reference Manual*, Reading: Addison-Wesley, 1990.

- If it is correct, argue why.
- If it is not correct, give a counterexample, and then state the correct rule as clearly and compactly as possible.

(d) Give an example similar to part (a) where pointers, not arrays, are used.

5. Scope vs. Lifetime

Scott text, Problem 3.4. Use 3 different programming languages in your 3 examples.

6. Criteria for Language Design

Illustrate each of the following "Criteria for Language Design", discussed so far during lecture:

(1) simplicity
(3) reliability
(4) fast translation
(5) efficient object code
(6) orthogonality
(7) language objects first-class
(8) transparent data types

For each criterion, give *one example for which it is satisfied, and one example for which it is violated.* Give concrete examples, not generalities, and do not repeat examples given in class. Choose your examples from among languages such as: APL, LISP, ML, Scheme, Icon, SNOBOL, C, C++, PASCAL, Modula2, ADA, Java, C#. Other languages mentioned in the text may be used. A partial list of language processors available in the Department is attached as an appendix below.

Give references to the language documentation or sources you use for your examples. Possible sources are: papers from the literature, the Scott text, references on library reserve, language reference manuals, and particular compilers/interpreters accessible to you in the Department or elswhere. In the latter case, be explicit about the language processor (e.g., "Sun Pascal 4.2 Compiler `pc`" not just "a Pascal compiler".) Translators available on home computers are fine, but cite them explicitly by manufacturer, version, date and hardware.

APPENDIX: Language Processors

```
Lectura (Linux)

C: cc, gcc, version 4.0.2, /usr/bin/gcc. (cc is a link to gcc)
C++: g++, version 4.0.2, /usr/bin/g++.
Icon: icont, icon, version 9.4.2,  /usr/local/bin/icon
java: version 1.5.0, /usr/local/bin/java
pascal: free pascal compiler 2.0.2, /usr/bin/fpc
prolog: GNU Prolog 1.2.16, /usr/bin/gprolog
scheme: umb-scheme, version 3.2, /usr/bin/umb-scheme
ml: objective ml 3.09.1, /usr/bin/ocaml
lisp: GNU CLISP 2.37, /usr/bin/clisp
Haskell, version 6.4.1, /usr/bin/ghc
Ruby, version 1.8.4, /usr/bin/ruby
Python, version 2.4.1, /usr/bin/python
perl, version 5.8.6, /usr/local/bin/perl
tcl, version 8.4.9, /usr/bin/tcl
```