Assignment II: Pseudocode Syntax

Hand in to instructor at beginning of class.

Pseudocode is a computing language which is designed to convey ideas, specifications, and algorithms. It eliminates as many implementation details as possible. In theory, a pseudocode program should be executable, given that the lower-level details are provided.

Design the syntax of a pseudocode programming language.

You will need to:

1. Select a small set of primitives for abstracting control, data, and names.
2. Decide upon a lexical form for your language primitives
3. Decide upon a syntax which structures how primitives are combined.
4. Use standard techniques to define acceptable lexical and syntactic forms.

Standard syntax specification techniques include task decomposition, regular languages, finite state acceptors, formal grammars, BNF and/or diagrammatic BNF.

Language primitives can be seen as addressing control, data, or naming. Control primitives are included in imperative languages to steer the course of program evaluation. Data primitives provide typing and abstraction. A language may provide a single data type, or several basic built-in types, or user-defined types. Naming primitives determine the binding of names to values, and the location of names and values in memory. Primitives that we have discussed in class include `loop`, `logic`, `let`, and `domain theories`. Others may include `sequence`, `order comparisons`, `subroutines`, `hierarchy`, and `i/o`.

Important:

1. The task is to think clearly and carefully about the meaning of the 19 *Principles of Programming Languages* (handed out in the first week). Check your design decisions for conformance to each of these principles.

2. You must *be explicit about the tasks* which your pseudocode is intended to address. Ask why each primitive and each structure is included in the design. What part of the task does each particular structure address?

3. Attempt to avoid structures which are intended to enhance implementation efficiency. Pseudocode is not intended to address implementation efficiency, rather it should maximize readability, comprehension, and absence of ambiguity.

4. The most difficult part of language design is minimizing interactions between primitives when they are combined.

Challenge: Implement a lexical scanner and syntactic parser for your language.