CS 240 Fall 2000
Data Structures and Data Management
Naomi Nishimura

Pseudocode

In the text and lectures, algorithms will often be expressed in *pseudocode*, a mixture of code and English (see Section 2.2 of the text, or for specific not necessarily good examples of particular pseudocodes, see pp. 4–5 of *Introduction to Algorithms* by Cormen, Leiserson, and Rivest, McGraw-Hill/MIT Press, 1990, *The Design and Analysis of Computer Algorithms* by Aho, Hopcroft, and Ullman, Addison-Wesley, 1974, *Computer Algorithms: Introduction to Design and Analysis* by Baase, 1978, and *Fundamentals of Computer Algorithms* by Horowitz and Sahni, Computer Science Press, 1984). While understanding pseudocode is usually not difficult, writing it can be a challenge.

Why use pseudocode at all? Pseudocode strikes a sometimes precarious balance between the understandability and informality of English and the precision of code. If we write an algorithm in English, the description may be at so high a level that it is difficult to analyze the algorithm and to transform it into code. If instead we write the algorithm in code, we have invested a lot of time in determining the details of an algorithm we may not choose to implement (as we typically wish to analyze algorithms BEFORE deciding which one to implement). The goal of writing pseudocode, then, is to provide a high-level description of an algorithm which facilitates analysis and eventual coding (should it be deemed to be a "good" algorithm) but at the same time suppresses many of the details that vanish with asymptotic notation. Finding the right level in the tradeoff between readability and precision can be tricky. If you have questions about the pseudocode you are writing on an assignment, please ask one of the course personnel to look it over and give you feedback (preferably before you hand it in so you can change it if necessary).

Just as a proof is written with a type of reader in mind (hence proofs in undergraduate textbooks tend to have more details than those in journal papers), algorithms written for different audiences may be written at different levels of detail. In assignments and exams for the course, you need to demonstrate your knowledge without obscuring the big picture with unneeded detail. Here are a few general guidelines for checking your pseudocode:

1. *Mimic good code and good English.* Using aspects of both systems means adhering to the style rules of both to some degree. It is still important that variable names be mnemonic, comments be included where useful, and English phrases be comprehensible (full sentences are usually not necessary).

2. *Ignore unnecessary details.* If you are worrying about the placement of commas, you are using too much detail. It is a good idea to use some convention to group

statements (`begin/end`, brackets, or whatever else is clear), but you shouldn't obsess about syntax.

3. *Don't belabour the obvious.* In many cases, the type of a variable is clear from context; unless it is critical that it is specified to be an integer or real, it is often unnecessary to make it explicit.

4. *Take advantage of programming shorthands.* Using `if-then-else` or looping structures is more concise than writing out the equivalent in English; general constructs that are not peculiar to a small number of languages are good candidates for use in pseudocode. Using parameters in specifying procedures is concise, clear, and accurate, and hence should not be omitted from pseudocode.

5. *Consider the context.* If you are writing an algorithm for quicksort, the statement `use quicksort to sort the values` is hiding too much detail; if we have already studied quicksort in class and later use it as a subroutine in another algorithm, the statement would be appropriate to use.

6. *Don't lose sight of the underlying model.* It should be possible to "see through" your pseudocode to the model below; if not (that is, you are not able to analyze the algorithm easily), it is written at too high a level.

7. *Check for balance.* If the pseudocode is hard for a person to read or difficult to translate into working code (or worse yet, both!), then something is wrong with the level of detail you have chosen to use.