Pseudocode Best Practices

Rather than continuing to **tell** you what you should and shouldn't do in pseudocode, we thought we'd try to let you determine for yourself (based on reading **our** pseudocode!) what is and isn't helpful in pseudocode.

Another helpful general hint: if you're looking for the general style we like in pseudocode, try looking at algorithms we include in assignment instructions and/or in sample solutions to assignment and worksheet problems. The point in pseudocode in general is to be understandable. That **is** our goal when we write algorithms for assignment instructions and sample solutions (hard as you may sometimes find that to believe...).

Pseudocode comparisons

Below we'll provide different ways to express the same algorithm, ranging from actual code to a couple of sentences. Read these over and try to understand what they do.

Type 1: Some real code

I do a lot work in MATLAB. So, ~~just to torture you~~ just for fun, here's an algorithm I implemented in MATLAB:

```
function Y = SomethingMysterious(X)

Y = {};
while length(X)
    w = X(1);
    c_w = 1;
    inds = [1];
    for i=2:length(X)
        if strcmp(X(i), w)
            c_w = c_w+1;
            inds = [inds i];
        end
    end
    Y{end+1} = {w, c_w};
    X(inds) = [];
end

end
```

1. How easy do you find it to **understand** what the algorithm in `SomethingMysterious` is doing?

   SOLUTION: Obviously there's no right or wrong answer here because the question asks for your opinion/experience. But I personally would not find that easy to understand, even though I know MATLAB.

2. If you needed to **analyze** `SomethingMysterious` (e.g., if we put this on an assignment and asked you to determine the running time), how easy would you find that to do?

SOLUTION: Again, this will vary by person. I could probably guess by looking at this that it's a worst-case $O(n^2)$ runtime, but wouldn't be able to tell much more than that without a fair bit of frustration.

Type 2: Slightly More Code-like pseudocode

Below I've taken `SomethingMysterious` and converted it to pseudocode. While pseudocode-writing style varies, this is a bit more detailed than I usually write.

```
PROCEDURE SomethingMysterious_v2(X):
    Y = [ ]
    while length(X) > 0:
        let w be the first element of X
        initialize the count c_w to 1
        initialize inds (indices to delete later) to the single-element list [1]
        for i=2:length(X)      // assume 1-based indexing
            if X[i] = w:
                c_w = c_w + 1
                append i to inds
        append {w, c_w} to Y
        delete from X all values at the indices stored in the array inds
```

1. How easy do you find it to **understand** what the algorithm in `SomethingMysterious_v2` is doing?

   SOLUTION: I don't know because I wrote it. But I think this is fairly clear and easy to follow, apart from the deliberately uninformative variable and function names!

2. If you needed to **analyze** `SomethingMysterious_v2` (e.g., if we put this on an assignment and asked you to determine the running time), how easy would you find that to do?

   SOLUTION: Again, see the answer above.

Type 3: Slightly More English-like pseudocode

This is a more high-level version of `SomethingMysterious`, which is more aligned with my personal preference for pseudocode-writing (though, again, there is some leeway for individual style/preferences here):

```
PROCEDURE SomethingMysterious_v3(X):
    Y = [ ]
    While X is not empty:
        Let w be the first element of X
        Count the number of occurrences of w in W (by scanning W from start to
           end) and call this result c_w
        Append w:c_w to Y
        Delete all occurrences of w from X
    return Y
```

1. Hopefully you've realized by now (or from one of the previous examples) that you have **already** been required to understand and analyze this function in quiz and assignment 1 (though we used more informative function and variable names in that case).

   Do you remember how easy or difficult it was to understand the algorithm when you first saw it? How do you think your experience might have varied if we had presented the algorithm in the style of `SomethingMysterious` or `SomethingMysterious_v2` instead?

SOLUTION: I'm just going to skip this one. I **did** include this for you to analyze in your quiz/assignment (after a fair bit of tweaking to try to hit just the right balance of detail and clarity), so I obviously think this is a reasonable way to present the algorithm.

You may prefer type 2 (the more code-like pseudocode) to this one, and that's entirely reasonable.

Type 4: Much More English-like pseudocode

Another alternative to actual pseudocode – which sometimes doesn't provide enough detail but is sometimes perfectly adequate – is to just describe your algorithm in plain English sentences. For `SomethingMysterious`, that might look like:

> Start with an empty bag of words. For each word in W, count the number of occurrences and append the word and its count (as a tuple) to the bag of words. Then delete all the occurrences of that word from W, and continue until W is empty.

1. Do you think this would have provided enough detail to help you understand and analyze the algorithm in quiz/assignment 1? Why or why not?

   SOLUTION: Your opinion may vary, but I consider this to be a bit lacking in detail for a question where you need to analyze the runtime. This may be easy enough to understand (though even then, I would consider type 3 to be more effective in this case), but it would require some assumptions to analyze its runtime.

   Again, in many instances, presenting your algorithm like this is perfectly fine (and we will do it ourselves in many assignment instructions and sample solutions). But in this particular case, I would go for slightly more "code"-ish style.

Pseudocode Best Practices

Discuss with your group what **you** think makes for "good" or "bad" pseudocode. Your TAs can provide guidance here too, and we'll provide some of our own thoughts in the sample solution for this exercise. But try to think of what you've learned from this exercise and from reading and writing pseudocode!

SOLUTION: here are a couple of suggestions, based on my own thoughts and those of the TAs:

- **Don't use language-specific syntax!** Part of the reason `SomethingMysterious` is so awful is that I (deliberately!) used some odd programmatic quirks that are particular to MATLAB. If your reader is not intimately familiar with the language you're using (i.e., has spent some time programming in it and done so **recently**), they will struggle to follow what you're doing. If your favourite language has a great built-in function, just describe what the function does rather than calling it in your pseudocode.

- **Avoid unnecessary code-related details.** For example, we suggest avoiding the use of curly braces or endif/endfor to end your conditional and loop statements. Your pseudocode will be easier to read if you indent clearly enough that someone can tell just by looking at it where the conditional/loop block ends. You also shouldn't use things like #include statements, import statements, etc.

- **Include enough detail that we could implement your algorithm ourselves, or analyze its runtime.** Your pseudocode should contain enough information that we could go program it ourselves – at the very least, we need to be able to check it against some test cases for correctness, which means we should be able to "run" it in our heads. It should also include enough detail for us to be able to analyze its runtime. That means that if you use a special-purpose data structure or a tailor-made helper function that makes your approach more efficient, you should describe it.

- **Brevity is good – provided you've given all the information you need to.** It's generally better to provide *shorter* pseudocode fragments that include all the information you need, than to provide needlessly long pseudocode. (This is in the same way that writing is easier to read when it's concise.) Figuring out how to be brief while keeping all the important information is a skill that takes practice (whether in writing or in pseudocode).

- **Treat pseudocode more like writing than like coding.** Always remember: your purpose in writing pseudocode is to be **easily understood by a person**. In that sense, writing good pseudocode is more like writing a paragraph than like writing actual code.

Common questions we get (on Piazza, etc.) are: can I use a (some particular well-known function) without actually implementing it? Can I use (some particular data structure) without describing it in detail?

Pretend you're a CPSC 320 TA, and a student asked you this sort of question. How do you think you might answer them?

SOLUTION: We'll state it again: your goal in writing pseudocode is to be easily understood. For purposes of this course, you should assume that your audience is someone with a CPSC 320-level of algorithms knowledge, but who isn't necessarily familiar with any particular programming language. So if you're describing a function or data structure that a typical CPSC 320 student will know about – sorting, a hash table, etc. – there's no need to describe its implementation in detail. (In fact, doing so will actually make your pseudocode less readable, as you'll be including unnecessary extra stuff that your reader will need to slog through.) But, if you're using an algorithm or data structure that is NOT standard for a student in CPSC 320, then you should explain (briefly and clearly) what it does.