Pseudocode

Pseudocode is code written for human understanding--not a compiler. You can think of pseudocode as “english code,” code that can be understood by anyone (not just a computer scientist). Pseudocode is not language specific, which means that given a block of pseudocode, you could convert it to Java, Python, C++, or whatever language you so desire.

Let us tell you now, pseudocode will be important to your future in Computer Science. Typically pseudocode is used to write a high-level outline of an algorithm. As you may already know, an algorithm is a series of steps that a program takes to complete a specific task. The algorithms you may (and will) be asked to write can get very complicated without a detailed plan, so writing pseudocode before actually coding will be very beneficial.

Goal: Practice writing pseudocode and understand how pseudocode translates to real code.

How to Write Pseudocode

There are no concrete rules that dictate how to write pseudocode, however, there are commonly accepted standards. A reader should be able to follow the pseudocode and hand-simulate (run through the code using paper and pencil) what is going to happen at each step. After writing pseudocode, you should be able to easily convert your pseudocode into any programming language you like.

We use indentation to delineate blocks of code, so it is clear which lines are inside of which method, loop, etc. Indentation is crucial to writing pseudocode. Java may not care if you don't indent inside your if statements, but a human reader would be completely lost without indentation cues. Remember: Human comprehension is the whole point of pseudocode.

So, what does pseudocode look like? Consider an algorithm to complete your math homework. Below we’ve included examples of both good and bad pseudocode.

Good Pseudocode	Real Code (in Java)	Bad Pseudocode

method doMathHomework():
  Get pencil
  Open textbook and notebook
 	
  Go through all the problems:
  Complete problem
  while the problem is
  wrong:
  Try again
 	
  Clean up your desk
  Submit your homework
	public void doMathHomework(){
	  this.getPencil();
	  _textBk.open();
	  _noteBk.open();
	
	  for(int i = 0; i < _problems.length(); i++){
	  _problems[i].solve();
	  while(!_problems[i].isRight()){
	  this.eraseSolution(i);
	  _problems[i].solve();
	
	  this.cleanDesk();
	  this.submit();
	}
		method doMathHomework():
		  Get things for homework
		
		  Do the problems correctly
		
		  Finish the homework

If we were provided the “bad pseudocode” and asked to convert it into Java, we would have to put a lot of thought into each line in order to convert it, which defeats the original purpose of writing it. When reading the bad pseudocode, a reader might (should) wonder, “what things do I need to do homework?” or “how do I do my homework correctly?;” this is a good indicator that your pseudocode isn’t specific enough.

The “good pseudocode” is detailed enough that in order to convert it into code, we simply had to read line-by-line and turn it into code. However, you’ll notice that every line is not in a 1:1 ratio with real code. For example, “Try again” is actually implemented as “this.eraseSolution(i); problems[i].solve();.” Because this was a small and simple implementation detail, we could omit it in the pseudocode for the sake of seeing the bigger picture. That said, the more specific your pseudocode is, the more useful it will be when you have to translate it into code--going too high-level will leave you with pseudocode like the “bad example.”

Also, it’s noteworthy that although I chose to convert my pseudocode into Java, I could have just as easily converted it into Python or C++; remember that pseudocode is *not* language specific so we are *not* looking for “almost Java” code, but instead, we are looking for a strong understanding of the algorithm at hand.

Differences in Pseudocode Style

In the example above, the pseudocode was very close to prose English. Yet we could still see how to translate it into object oriented code. In some cases, however, the pseudocode could very much resemble code. Consider an algorithm that prints out all odd numbers between 1 and 10:

```
method printOddNumbers():
      i = 1
      while i < 11:
            if i is not divisible by 2:
                  print i
            i++
```

This pseudocode style was much more appropriate for the given algorithm, and it wouldn’t make much sense to have tried to make it better resemble prose English. Pseudocode style depends greatly on personal preference, the problem you’re asked to solve, and on what you're optimizing for: speed, readability, aesthetics, etc. As you write more pseudocode, you will discover what style comes most naturally to you.

In summary:

> Roses are red, violets are blue, pseudocode comes in many shapes and sizes, just make sure it appropriately outlines the algorithm you're writing.

Drawing Turtles

Remember the turtle demo from lecture? You are going to create your own! Your turtle can do the following things:

- move forward a number of steps (each step is one pixel long)
- turn left 90 degrees
- turn right 90 degrees
- put its pen down, which starts drawing the turtle's path
- lift its pen up, which stops drawing the turtle's path

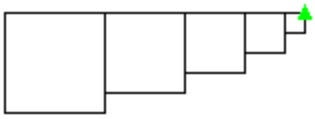
The width of the turtle's pen is one pixel. The initial direction of the turtle is facing north.

Check Point 1: Pseudocode

You will be writing pseudocode for an algorithm that directs the turtle to draw a chain o squares decreasing in size by 10 pixels.

Hint: the side length of a square should never be less than 0, right?

Example: The chain should look like this for a starting length of 50:



- Fill in the following pseudocode method in Sublime or another text editor:

```
/* inputs:
 *   turtle - turtle that is drawing
 *   sqSideLength - initial square's length
 */
method drawSqChain(turtle, sqSideLength){
     // fill me in with pseudocode here!
}
```

- **Show your pseudocode to a TA before moving on.**

Now that you have finished writing your pseudocode, put your skills to the test and go code your algorithm. If you have written your pseudocode well, it should be a trivial step to translate it line by line into Java.

Check Point 2: Coding

- Run `cs015_install labPseudocode`. This will install stencil code in /home/<yourlogin>/course/cs015/labPseudocode/.
- Open up eclipse, click App.java and click "Run as… >> Java Application"
- What's this?! You should notice that your program will not run…
- Continue reading to see how to fix this.

Oh no! It appears that Spy Turtles have locked the Turtle Drawer, which is preventing you from being able to execute your code. If you inspect the code in `MyTurtleDrawer.java` you'll notice that the program can be unlocked by successfully passing in a password (that is dependent on your username) to the `checkPassword(String username, String password)` method. Due to the cunning of the Spy Turtles, our human eye is unable to see the inner workings of this method. So this looks like a job for… the Eclipse Debugger (and your super spy abilities).

Your mission: Unlock this program.

Check Point 3: Mission Impossible

`checkPassword(String username, String password)` works by using your login to generate a password. It then stores this in the variable `correctPassword` and checks whether your inputted password equals `correctPassword`.

To unlock `MyTurtleDrawer` you have to figure out what this password is! You *could*

try to figure out how `checkPassword(...)` generates the password, but *this is pretty hard*.

Instead, use the Eclipse Debugger to find the value of `correctPassword`!
(See Lab Debugging Part 2 for a refresher on the Debugger)

Hint: We should use the Debugger to see the **inner** workings of the method.

The program has been unlocked and the Spy Turtles' plans have been foiled! Great! Now let's *actually* get coding.

Check Point 4: Actually Coding

- Fill in the `drawSqChain(Turtle turtle, int sideLength)` method in the `MyTurtleDrawer.java` file. Look at the comments in the file to see what methods of Turtle you can call.

  Note: You do not need to modify the other file, `App.java`
  Note: The Turtle draws from a random point on the screen

- If your turtle does not draw as expected, go back and hand simulate your pseudocode again! What is going wrong?

- **Show your program to a TA to get checked off for today's lab!**

Congratulations on finishing the lab. If you are finished early, feel free to play around with the drawing turtles and make some cool designs (maybe try a spiral or a filled in square); this is a great way to practice writing loops.