**Programming Guidelines**

**Robustness**

Your primary task is to create a working, robust program. This means that your program should produce correct answers on all legal input and produce comprehensible error messages on invalid input. Keep in mind that unreasonably long running time is probably an error (unless otherwise mentioned). How should you ensure robustness? At the least, try to complete these steps before submitting your code:

* Compile without warnings or errors. (Notify me of errors in assignment code.)
* Run correctly on all test data given you. (Tell me if you find an error in test data.)
* Run correctly on test data of your own which has:
  + Difficult cases
  + Boundary cases
  + Minorly and egregiously incorrect input
  + The empty file

This doesn't guarantee robustness, but it's a good start. Turn in your test cases; they may help us grade your work better.

**Coding Clearly**

You should *always* make an effort to write code that is easy for other people to read. In the real world, this is an important skill that is valued higher than cleverness or using the least number of lines/characters possible. In this class, it is important so that your program can be understood so they can be graded fairly. By reading through your code and comments, one should be able to figure out how your code is organized and why it works (or fails). If you cannot, you will likely have points deducted.

*Commenting:*  One aspect of writing clear code involves commenting your code so that it is clear what it does without reading it line by line. Comments should be used: (a) at the top of each file to tell what code it contains, (b) at the top of each method to tell what it does, (c) at the declaration point of important variables, (d) on any lines whose purpose is non-obvious.

*Variable names:* Variable names should conform with Java conventions (e.g., myVariableName, MyClassName, MY\_CONSTANT\_NAME) and should give a good idea of what purpose that variable serves in your program.

*Coding simply:* Although Java allows it, there is no benefit to writing overly complicated statements such as:

if ((i = myfunc(counter++)) < num\_iterations)

{

...

}

when they could easily be rewritten in a clearer manner:

i = myfunc(counter);

counter++;

if (i < num\_iterations)

{

...

}

Coding should not involve grandstanding or distracting cleverness. It should be like writing English sentences that you truly want someone to understand (including yourself, a few months later).

**Speed**

As long as your program takes a reasonable amount of time and your data structures and algorithms have the correct time and space complexity, we are not interested in small constant-factor differences. **Robustness and clarity come first!**

**Using Java Class Libraries**

Unless specifically approved or directed to, you should **not** use Java class libraries in your programming assignments to implement data structures and algorithms that are your responsibility. This rule is admittedly in contrast to many situations in the real world, but a large goal of this course is to fully understand core data structures that are so widely useful that Java provides them in its standard library. Often such understanding comes from implementing them yourself.