Coding, a 213 approach

Before you Start
- go to classes
- go to recitation
- read the textbook
  - before the lecture, or no later than 24 hrs afterwards
  - if you miss any, go over the lecture slides with a classmate

Print out Assignment/Specification
- !!!we mean it!!!
- you can annotate it

Gathering Information

Read handout once
- also re-read relevant parts of textbook

Read handout again
- what functionality will you need?
- put boxes around the things you don't understand
- look for over arching concerns (is runtime really important? source size? etc)
- what are you *not* allowed to do?
- write down your questions on the printout

Read given code/existing code (if applicable)
- don’t reinvent the wheel.
- note (on your print out) things you need to know to use provided code.
- *if* there is testing code provided, pay particular attention to what it *doesn’t* cover

Make a checklist of what your code needs to do
- what support code will you need?
- what is being asked?
- what can happen out of order?

Sketch out a design for your program
- try to turn your checklist into pseudo-code outline
- what data structures/algorithms could help?
  - what are the drawbacks?
  - Ockham's/Occam's razor

Make an outline of your Testing Plan
- what tools can help you
- debuggers
  - what debugger features will you use
- what test code will you write?
- what race conditions are likely to show up
  - and how will you know?

Write down some execution stories
- For example:
  1. The user types "/bin/sleep 10 &".
  2. The shell forks.
3. The child ... 
4. The shell ... 
5. Eventually, ... 
6. The shell ... 
   - Using your outlines, will your code agree with the story?

Read your Assignment again 
   - does your pseudo code outline violate spec? 
   - does your testing plan violate any policies?

Repeat Steps as Necessary

Programming/Debugging

If you get stuck 
   - bring your outlines, printout, & checklist to a TA 
   - go back to your stories, using debugging, where does your code 
     diverge from your story 
   - Course staff can help you best with specific, describable problems, but cannot write code for you or debug 'It just doesn't work' 
     - emphasis on "describable" 
     - the more documentation you have on your bug, the better it will be

Write your code and test code together 
   - not in the same file, but similar times

comment your code as you go 
   - so you know what you were thinking when you work on it again

Before Turning Project in
review code/comments 
   - remove inane, old, or useless comments 
   - make sure all functions have block comments at the top explaining what they do, and special algorithms, etc

document any standing bugs 
   - TAs reserve the right to be lenient if we can see you were on the right track