How to Develop Small Programming Projects*

Jill Seaman
CS 1428
Fall 2019

*without banging your head against the wall

Getting Started

- Start early: we always underestimate the complexity of the problem.
- Understand the material: study first!
- Understand the requirements (READ the directions, don't make assumptions).
- Use some top-down design to break up the problem into pieces.
- Make a plan before you implement.

Develop Programs Progressively (incremental development)

- Do not attempt to implement an entire program all at once.
- Implement a very small, but workable, part.
- Compile, fix syntax errors, execute (test), debug
- Add another small part, refine the code
- Compile + test. Any new errors are (probably) due to newly added code.
- Repeat until complete

Compiler (syntax) Errors

- Fix only the first one or two before re-compiling, later errors may be dependent.
- Don't speak compiler?
  Google the error text (with caution)
- Think of common syntax errors
  - Missing semicolons
  - Misspelled variable names
  - Misplaced ( ) or { }, backwards << or >>

Testing

- Testing: running the program with simulated data, checking the actual output against expected output, in order to find bugs
- Bug: coding mistake causing an error in output
- Test Case: a set of specific input data and the corresponding expected program output
- Choose input data wisely:
  - Values used in if/while conditions
  - Smallest and largest valid values of a dataset
  - Put data in multiple positions: for maximum, put max value in first position, then last position, then middle position

Debugging

- Test failure: actual output from running a test case does not match the expected output.
- Debugging: figure out why it failed, find the coding mistake and fix it.
- Add output statements in strategic places:
  - cout the values of variables (label them!)
  - trace execution path, see which statements are being reached. Add `cout<<"here1"<<endl;` statements periodically in your program.