CS301

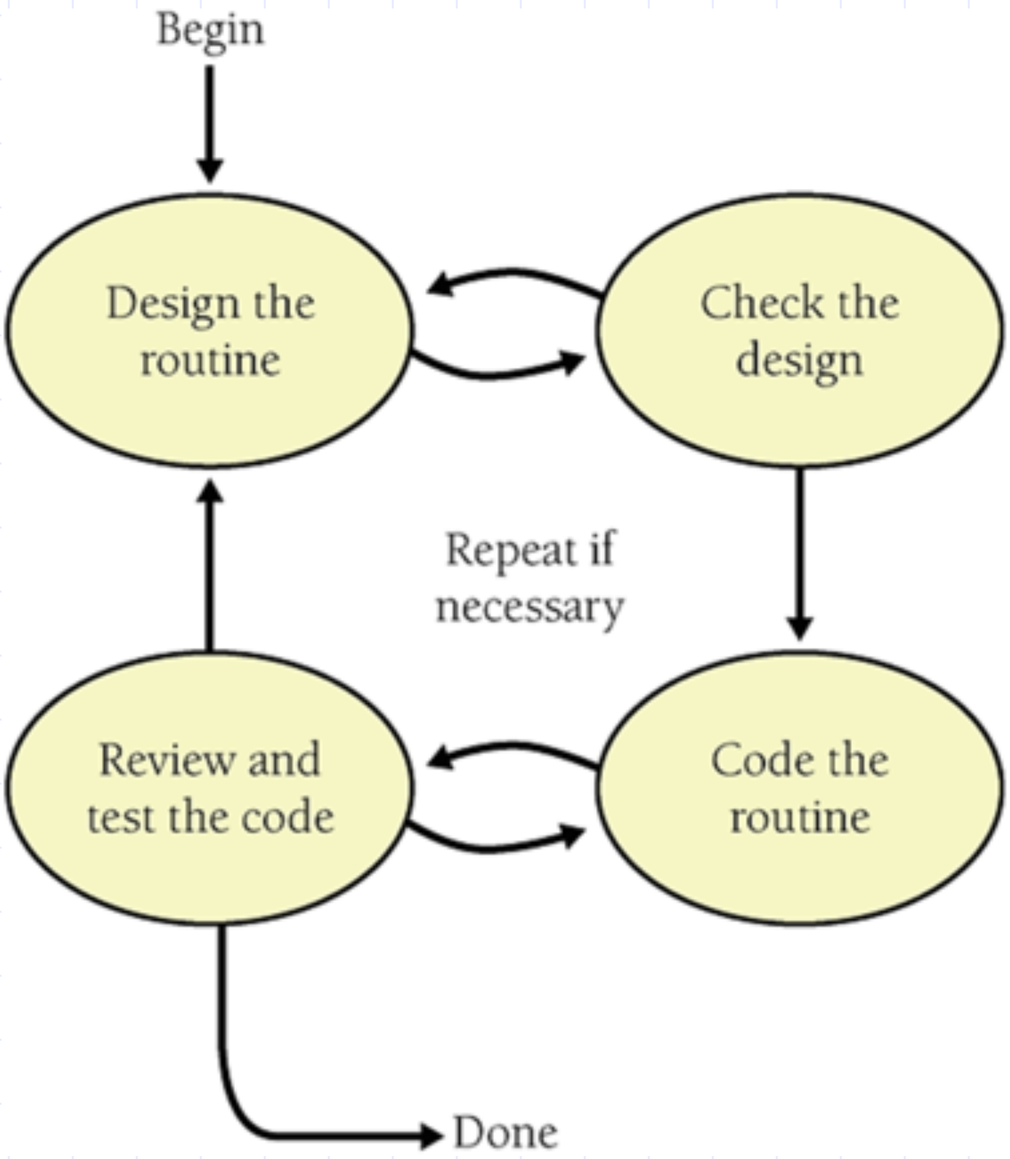
Peter Kemper

R 104A, phone 221-3462, email:kemper@cs.wm.edu

The Pseudocode Programming Process

Reference: Steve McConnell, Code Complete, 2n ed
Chapter II.9

Steps in Building a Method/Routine



Most methods simple, straightforward

For complicated ones:

- needs thinking
- have choices

Necessary:

- purpose/responsibility
- data & algorithm
- input & input constraints
- output & its constraints
- shared data

Pseudocode

- Key idea: write up how the method works in pseudocode
  - Think it through, do not forget cases, clarify I/O
  - Does not interfere with coding issues
  - Once written, write code for it, (no need for comments)
- Pseudocode
  - Use English-like statements that precisely describe specific operations
  - Avoid syntactic elements from the target programming language. Use a slightly higher level of abstraction!
  - Write pseudocode at the level of intent. Describe the meaning of the approach
  - Write pseudocode at a low enough level that generating code from it will be nearly automatic.
    - If level is too high -> refine into details

Benefits of PPP

- Pseudocode makes reviews easier.
- Pseudocode supports the idea of iterative refinement.
- Pseudocode makes changes easier. (before coding)
- Pseudocode minimizes commenting effort.
- Pseudocode is easier to maintain than other forms of design documentation.

- Key point:
  - As a tool for detailed design, pseudocode is great!

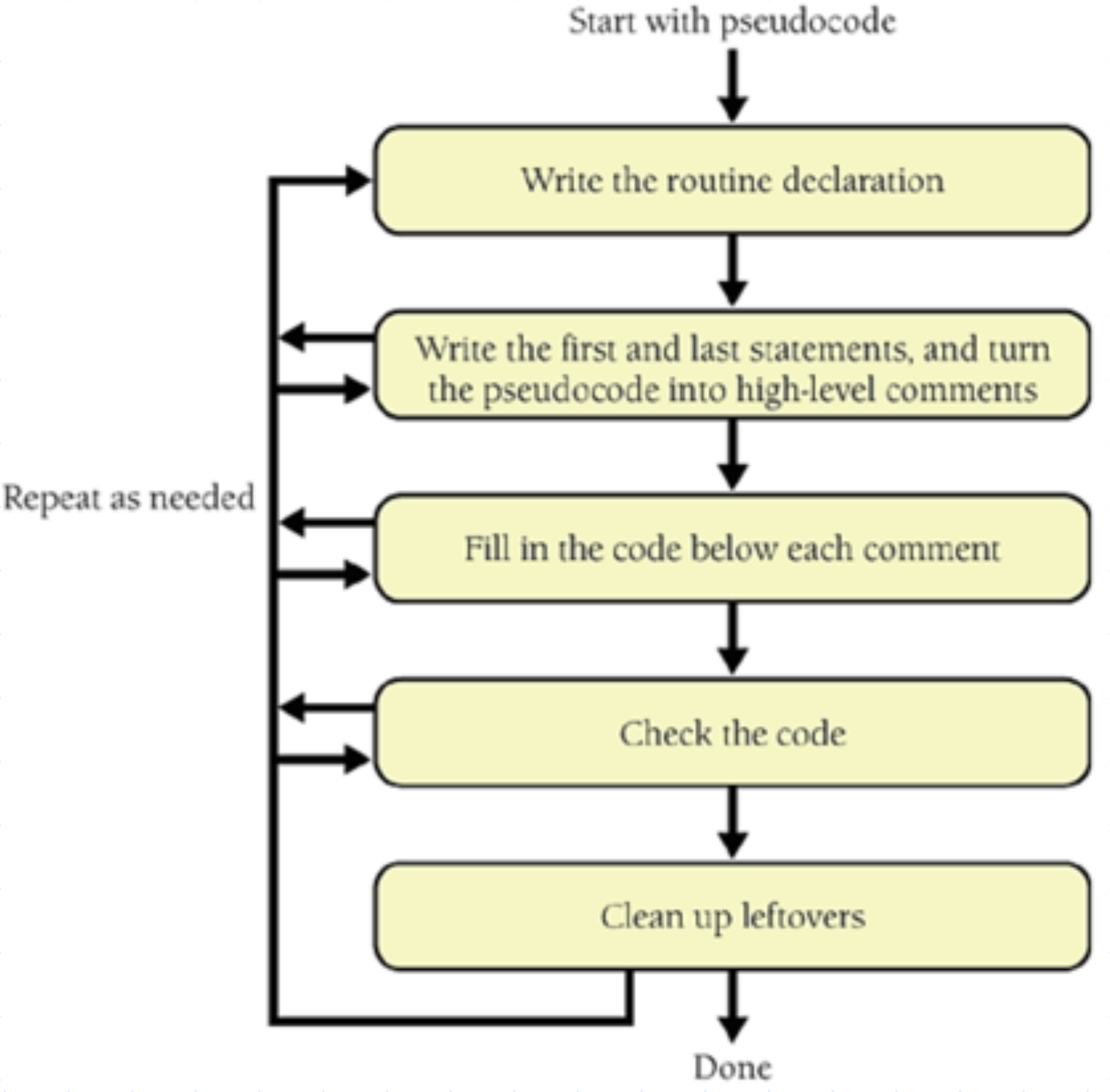
High-Level Method Design contains

- The responsibility/purpose of the routine
- The information the routine will hide
- Inputs to the routine
- Outputs from the routine
- Preconditions that are guaranteed to be true before the routine is called
- Postconditions that the routine guarantees will be true before it passes control back to the caller

Pseudocode programming process

- Starting point: Given design
- Step 1: Check prerequisites (job is well-defined, required)
- Step 2: Name the routine
- Step 3: Decide how to test the routine
- Step 4: Check for existing solutions (reuse code, ideas)
- Step 5: Consider error handling
- Step 6: Consider efficiency (no premature optimization)
- Step 7: Check for existing data types, algorithms
- Step 8: Think about the data (data types, what to store)
- Step 9: Write/refine description in pseudo code
- Step 10: Check/review pseudo code
    - yourself, ask someone else
    - make sure everything is clear to you before you start coding
- Iterate: try a few ideas and keep the best!

Once you got the pseudo code


Start with pseudocode
Write the routine declaration
Write the first and last statements, and turn the pseudocode into high-level comments
Repeat as needed
Fill in the code below each comment
Check the code
Clean up leftovers
Done

Checklist

- Have you checked that the prerequisites have been satisfied?
- Have you defined the problem that the class will solve?
- Is the high-level design clear enough to give the class and each of its routines a good name?
- Have you thought about how to test the class, each of its routines?
- Have you thought about efficiency mainly in terms of stable interfaces and readable implementations or mainly in terms of meeting resource and speed budgets?
- Have you checked the standard libraries and other code libraries for applicable routines or components?
- Have you checked reference books for helpful algorithms?
- Have you designed each routine by using detailed pseudocode?
- Have you mentally checked the pseudocode? Is it easy to understand?
- Have you paid attention to warnings that would send you back to design (use of global data, operations that seem better suited to another class or another routine, and so on)?

Checklist

- Did you translate the pseudocode to code accurately?
- Did you apply the PPP recursively, breaking routines into smaller routines when needed?
- Did you document assumptions as you made them?
- Did you remove comments that turned out to be redundant?
- Have you chosen the best of several iterations, rather than merely stopping after your first iteration?
- Do you thoroughly understand your code? Is it easy to understand?