CS301 Software Development

Instructor: Peter Kemper

The Pseudocode Programming Process

Example Task

- Class SlidingPuzzle

Figure: en:User:Booyabazooka - http://en.wikipedia.org/wiki/Image:15-puzzle.svg

- Method

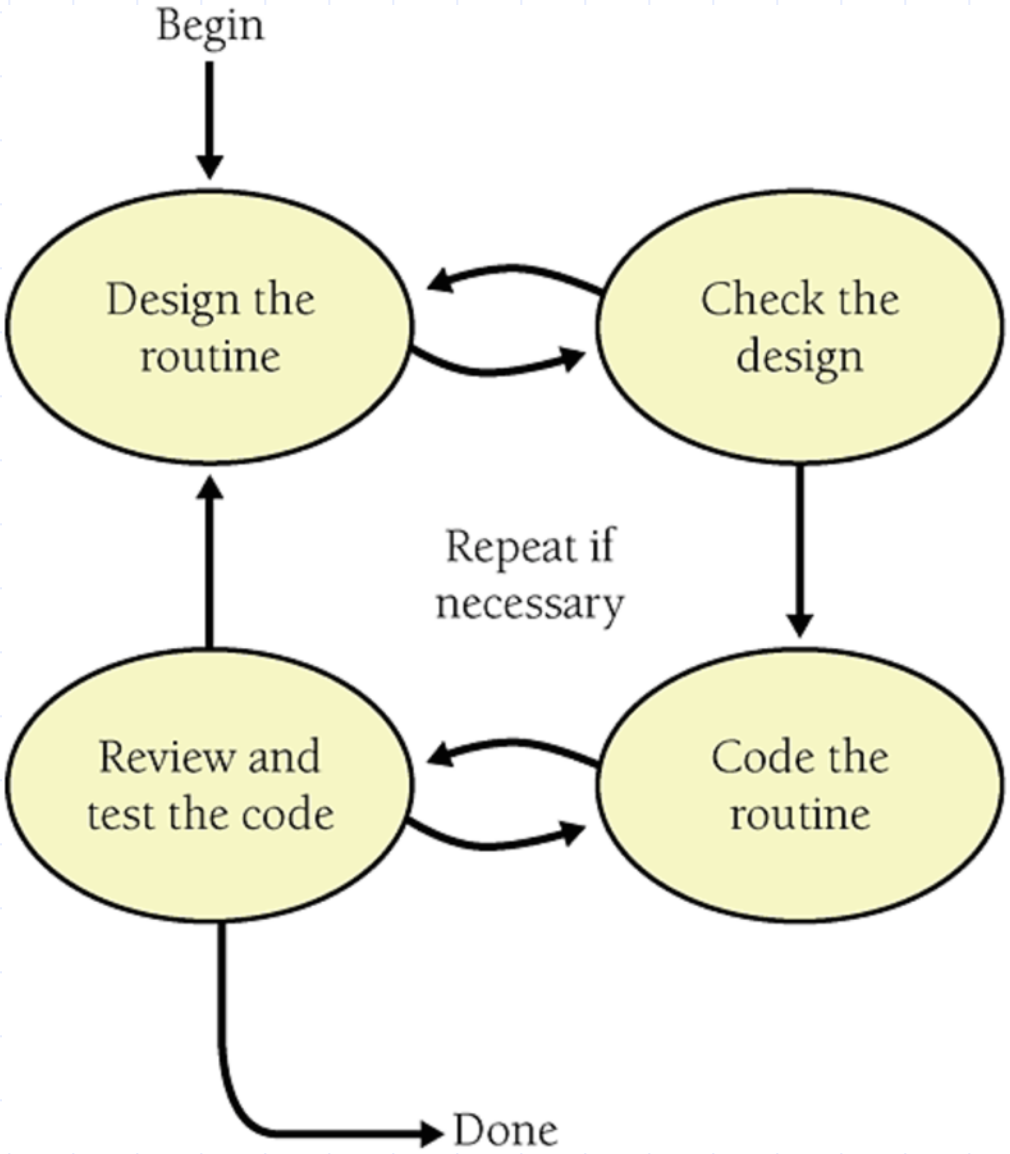
  private boolean searchBreadthFirst()

- Description

  "Method finds a solution to a given sliding puzzle problem by applying Breadth-First Search. It returns true if it finds a solution, false if not. A computed path to the solution should be stored in a private instance variable."

- How do you come up with an implementation?
  - Start: understand problem & algorithm
  - Then: …

Steps in Building a Method/Routine



Most methods simple, straightforward

For complicated ones:
- needs thinking
- come with choices

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

Figure: S. McConnell, Code Complete, 2nd Ed.

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

Low-Level Method Design

- Key idea: separate detailed design from coding
- Design:
  - 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
- Pseudocode => comments, no need for more
- In other words:

"Write comments first!"

Pseudocode

- What is Pseudocode?
  - English-like statements that describe specific operations

On Writing Pseudocode

- Write pseudocode at level of intent; describe meaning of approach
- Choose level of abstraction
  - High enough for human to understand solution
  - Low enough such that coding is last step of refinement
- Avoid syntactic elements of a programming language.
- Example:
  - Good: “find matching element in data set”
  - Bad: “increment i and compare a[i] with v till both are equal then return”

Benefits of Writing Pseudocode

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

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

Pseudocode Programming Process (PPP)

- Starting point: Given high-level 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

Pseudocode Programming Process (PPP)

- 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 pseudocode
- Step 10: Check/review pseudocode
  - yourself, ask someone else
  - make sure everything is clear to you before you start coding
- Iterate: try a few ideas and keep the best!

Coding: Once you got the pseudocode …

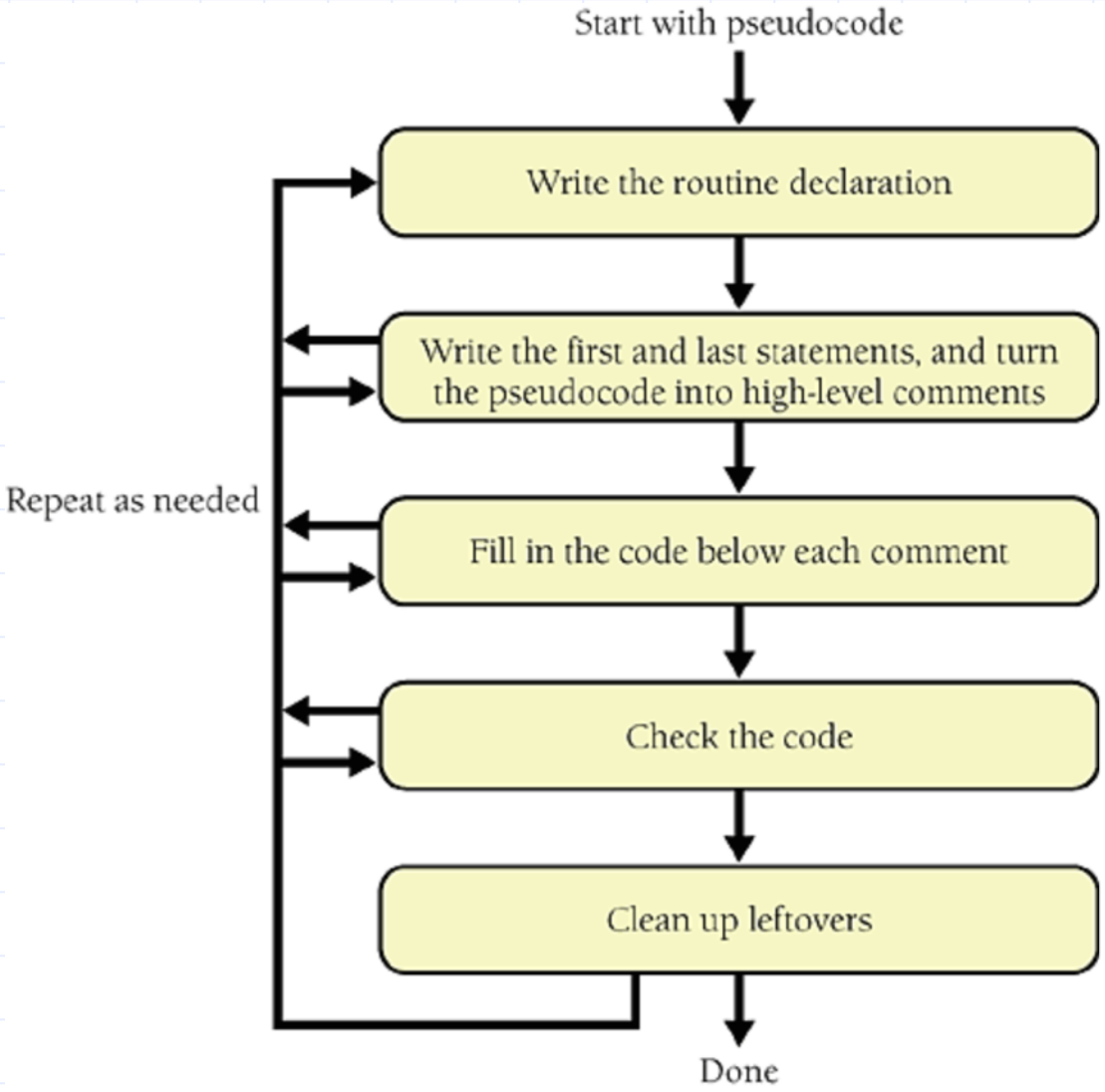


Figure: S. McConnell, Code Complete, 2nd Ed.

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?

Checklist (cont.)

- ✓ 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 (cont.)

- ✓ 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?

Summary

- The bottom line for PPP:
  - Think it through before you code
  - Write down your thoughts in pseudo code
  - Review
  - Code
  - Review again