"Pseudocode Programming Process"

↓

Pseudocode, the PPP and Alternatives

Noah Doersing

noah.doersing@student.uni-tuebingen.de

June 10, 2015

Overview

1. Why Write Pseudocode?

- Once a class has been designed and the major routines have been drafted, their high-level descriptions must be translated into code.
- Routine creation benefits from a systematic approach: the *Pseudocode Programming Process* (PPP).
- Enables checking of a routine design before implementation, making many kinds of errors apparent before any code is written.
- **Pseudocode is an effective way of iteratively creating and testing a routine.**

2. Good Pseudocode

- Use precise, natural-language descriptions of specific actions.
- Avoid code-like statements. Pseudocode should be at a higher level than code in order to avoid getting restricted by characteristics of a particular programming language.
- Capture intent: Describe what the design is doing, not how it's going to be implemented later on.
- Don't settle for the first approach you think of. Pseudocode allows you to evaluate multiple designs or variations on a design with little effort.
- Iterate: Refine your Pseudocode until it would be easier to just write code instead.

Example (Good Pseudocode)

```
Keep track of current number of resources in use
If another resource is available
    Allocate a dialog box structure
    If a dialog box structure could be allocated
        Note that one more resource is in use
        Initialize the resource
        Store the resource number at the location provided by
          the caller
    Endif
Endif
Return true if a new resource was created; else return false
```

Steve McConnell: Code Complete, Second Edition. Microsoft Press, 2004, page 219.

Example (Bad Pseudocode)

```
increment resource number by 1
allocate a dlg struct using malloc
if malloc() returns NULL then return 1
invoke OSrsrc_init to initialize a resource for the
  operating system
*hRsrcPtr = resource number
return 0
```

Steve McConnell: Code Complete, Second Edition. Microsoft Press, 2004, page 218.

3. Benefits of the PPP

- Permits (not only initial) review of complex designs, reducing
  - the number of errors accumulating over time,
  - the need to review the code itself.
- Pseudocode can be kept as inline comments, reducing commenting effort.
- This serves as easily maintainable documentation: code and comments are in the same place, and as such easier to keep in "sync".
- Iterative refinement: Refine the high-level design to pseudocode, then refine the pseudocode to code.
  $\Rightarrow$ Different kinds of errors are caught at different stages.

4. Example: Constructing a Routine with the PPP

First step: Design

Example (Informal spec of the routine)

`listFinalGrades()` takes a year as an input argument, reads the presentation, term paper and review grades for every student taking the SCT course in that year from a database and returns a list of all students with their final grade.

- Check prerequisites: is the routine well defined, necessary, and does it fit into the class design?
- Define inputs, outputs, preconditions, postconditions.
- Name the routine.
- How are errors handled?

4. Example: Constructing a Routine with the PPP

Second step: Pseudocode (1)

- Start with a high-level description that is general enough to be used as a header comment, and begin writing pseudocode.

Example (High-level description of the routine)

This routine returns a list of all students taking the SCT course in the given year, along with their final grades.

Example (High-level pseudocode)

```
validate the year
retrieve a list of students from the database
compute the final grades
return the names and grades as a list
```

4. Example: Constructing a Routine with the PPP

Second step: Pseudocode (2)

- Iteratively refine and check the pseudocode, adding more detail until it would be easier to just write code instead.

Example (Mid-level pseudocode)

```
if the year is valid
    connect to the database and retrieve info about all students
      in the given year
    keep a list of the students and their final grades
    for all students
        calculate a weighted average of the three grades
        add their name and final grade to the list
    endfor
endif
return the list; or return false if no list could be generated
  for the given year
```

4. Example: Constructing a Routine with the PPP

Third step: Code (1)

Example (After adding the header comment and routine interface)

```
/**
 * This routine returns a list of all students taking the SCT course in
 * the given year, along with their final grades.
 */

public List<Pair<String, Float>> listFinalGrades(int year) {
    // if the year is valid
        // connect to the database and retrieve info about all students
             in the given year
        // keep a list of the students and their final grades
        // for all students
            // calculate a weighted average of the three grades
            // add their name and final grade to the list
        // endfor
    // endif
    // return the list; or return false if no list could be generated
         for the given year
}
```

Example (After filling in the source code below each comment)

```
/**
 * This routine returns a list of all students taking the SCT course in
 * the given year, along with their final grades.
 */

public List<Pair<String, Float>> listFinalGrades(int year) {
    // if the year is valid
    if (validateYear(year)) {
        // connect to the database and retrieve info about all students in the given year
        [...] Student[] students = [...]

        // keep a list of the students and their final grades
        List list = new ArrayList<Pair<String, Float>>();

        // for all students
        for (Student student : students) {
            // calculate a weighted average of the three grades
            float finalGrade = 0.4 * student.presentationGrade
                             + 0.4 * student.termpaperGrade
                             + 0.2 * student.reviewGrade;

            // add their name and final grade to the list
            list.add(new Pair<String, Float>(student.name, finalGrade));
        }

        // return the list; or return false if no list could be generated for the given year
        return list;
    }
    return false;
}
```

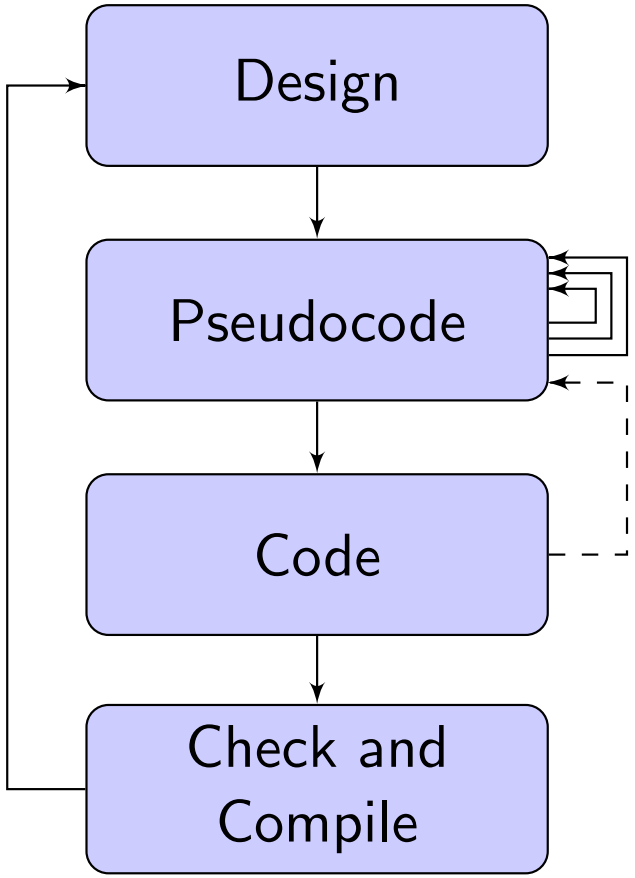
4. Example: Constructing a Routine with the PPP

Fourth step: Check and Compile

- Check whether to move some of the code into a separate routine, and apply the PPP recursively.
- Mentally execute each path and check for errors, thereby making sure you fully understand the routine.
- Compile the routine.
- Test and debug the routine.
- Check for and address "code smells" and other undesirable characteristics.
- Remove comments that have been necessary during the pseudocode stage, but are now redundant.

4. Example: Constructing a Routine with the PPP
Summary

- *Design* the routine:
  check prerequisites, inputs, outputs, error handling, etc.
- Iteratively write *pseudocode* for the routine, as well as a high-level description.
- Convert the pseudocode to comments, add source *code*.
- *Check and compile* the routine.



5. Alternatives to the PPP (1)

(Rapid) Prototyping

- Write a prototype (in a higher-level language or with specialized tools) in order to evaluate the effectiveness of an approach and catch mid-level errors.
- After addressing potential issues, reimplement the routine in the target programming language.

"Build One To Throw Away"

Prototyping in the target programming language:
Second implementation almost always better than the first.

5. Alternatives to the PPP (2)

Refactoring

After the initial implementation of a routine, methodically eliminate "code smells", thereby improving code quality. *(CC Ch. 24)*

Test-driven Development

Construct routines by writing test cases before code.
(Presentation "Testing" on July 1; CC Ch. 22; PP Ch. 8, Sec. 43)

Design by Contract

Define routines based on their preconditions and postconditions.
(Presentation "Defensive Programming" today; CC Ch. 8)

6. Key Points

Pseudocode is a tool for writing routines that

- enables the programmer to iteratively refine the routine,
- facilitates the detection of mid-level errors before writing code,
- and produces easily maintainable inline documentation.

Some of the same goals can be achieved with *Prototyping*, *Refactoring*, *Test-driven development* or *Design by Contract*.

Thank you for your time and attention!