

Pseudocode For Pros

https://make.sc/spd1.4
April 3, 2019

Guidelines / How To

What is Pseudocode?

An informal, English-like notation for describing how an algorithm, a routine, a class, or a program will work.

Step 1: Make a Plan

Write english-like statements that precisely describe specific operations your code will run.

Step 2: Code the Plan

Build the code around the pseudocode!

No Code Zone

Avoid syntactic elements from the target programming language.

Describe Intent

Write at the level of intent.

Describe the meaning of the approach rather than how the approach will be implemented in the target language.

Nuts & Bolts

Write at a **low enough level** that the act of writing your code from it is nearly automatic.

Comments for Free

Turn it into comments in your code Eliminates most commenting effort.

If the pseudocode follows the guidelines, the comments will be complete and meaningful.

Building a Routine

Step 1: Design

Design the Routine

Step 2: Verify

Check the Design

Step 3: Write Code

Code the Routine

Step 4: Review & Test

Review & Test the Code

Building a Class

Step 1: Design

Create general design for the class.

Step 2: Construct

Construct each routine within the class.

Step 3: Review & Test

Review/test the whole class.

9.1 Summary of Steps in Building Classes and Routines

Class construction can be approached from numerous directions, but usually it's an iterative process of creating a general design for the class, enumerating specific routines within the class, constructing specific routines, and checking class construction as a whole. As Figure 9-1 suggests, class creation can be a messy process for all the reasons that design is a messy process (reasons that are described in Section 5.1, “Design Challenges”).

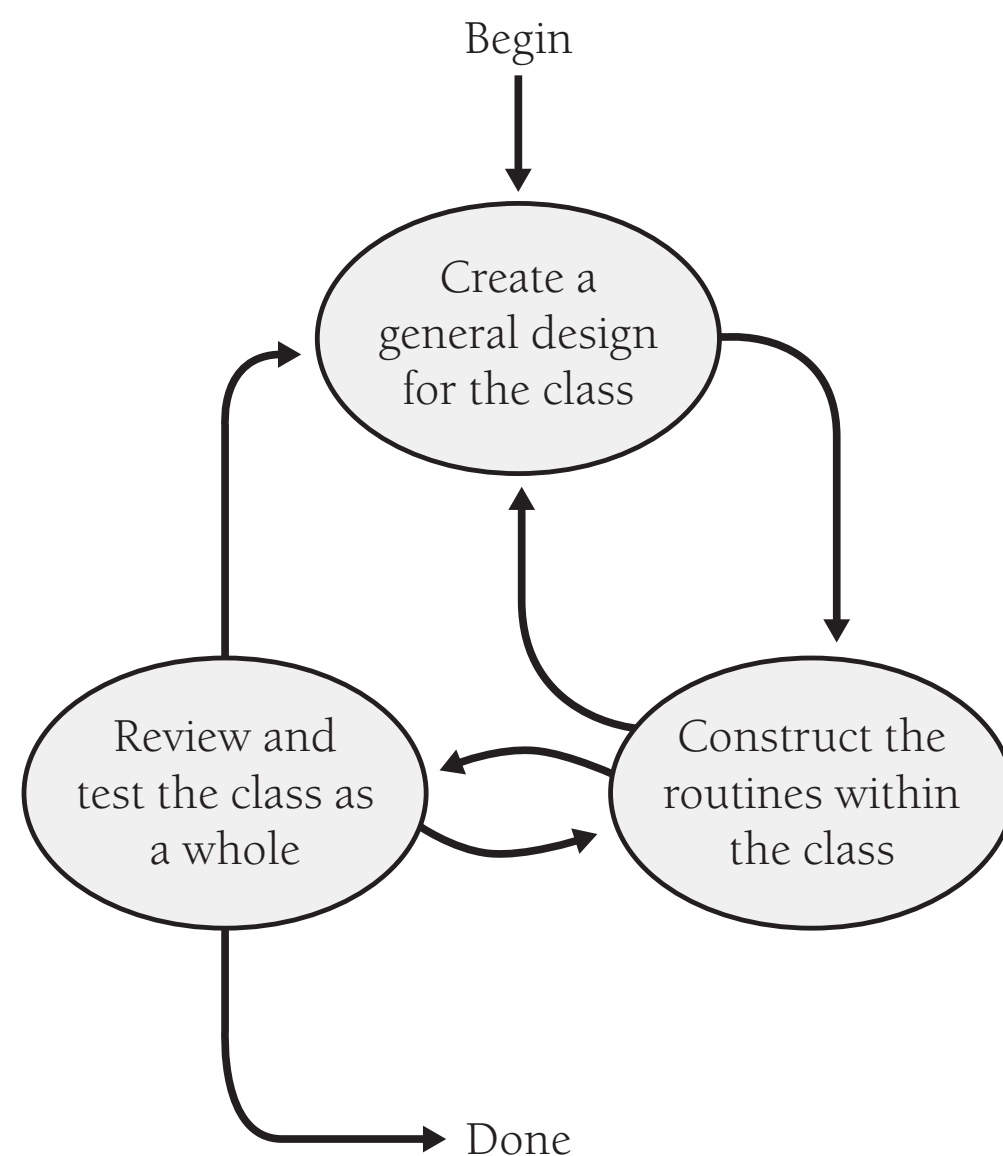



Figure 9-1 Details of class construction vary, but the activities generally occur in the order shown here.

Steps in Creating a Class

The key steps in constructing a class are:

Create a general design for the class  Class design includes numerous specific issues. Define the class's specific responsibilities, define what “secrets” the class will hide, and define exactly what abstraction the class interface will capture. Determine whether the class will be derived from another class and whether other classes will be allowed to derive from it. Identify the class's key public methods, and identify and design any non-trivial data members used by the class. Iterate through these topics as many times as needed to create a straightforward design for the routine. These considerations and many others are discussed in more detail in Chapter 6, “Working Classes.”

Construct each routine within the class Once you've identified the class's major routines in the first step, you must construct each specific routine. Construction of each routine typically unearths the need for additional routines, both minor and major, and issues arising from creating those additional routines often ripple back to the overall class design.

Review and test the class as a whole Normally, each routine is tested as it's created. After the class as a whole becomes operational, the class as a whole should be reviewed and tested for any issues that can't be tested at the individual-routine level.

Steps in Building a Routine

Many of a class's routines will be simple and straightforward to implement: accessor routines, pass-throughs to other objects' routines, and the like. Implementation of other routines will be more complicated, and creation of those routines benefits from a systematic approach. The major activities involved in creating a routine—designing the routine, checking the design, coding the routine, and checking the code—are typically performed in the order shown in Figure 9-2.

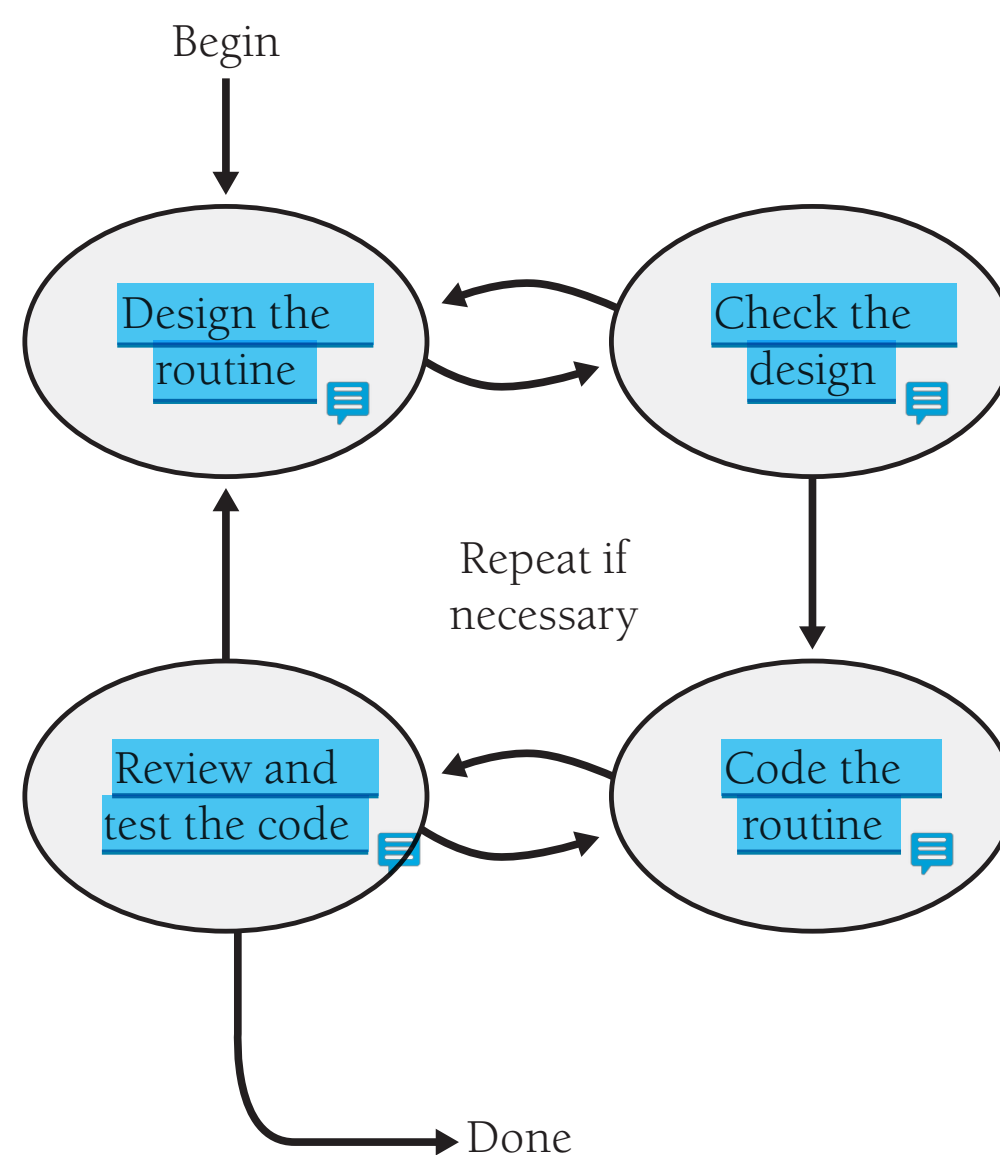


Figure 9-2 These are the major activities that go into constructing a routine. They're usually performed in the order shown.

Experts have developed numerous approaches to creating routines, and my favorite approach is the Pseudocode Programming Process, described in the next section.

9.2 Pseudocode for Pros

The term “pseudocode” refers to an informal, English-like notation for describing how an algorithm, a routine, a class, or a program will work. The Pseudocode Programming Process defines a specific approach to using pseudocode to streamline the creation of code within routines.

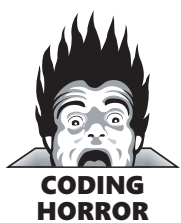
Because pseudocode resembles English, it’s natural to assume that any English-like description that collects your thoughts will have roughly the same effect as any other. In practice, you’ll find that some styles of pseudocode are more useful than others. Here are guidelines for using pseudocode effectively:

- Use English-like statements that precisely describe specific operations.
- Avoid syntactic elements from the target programming language. Pseudocode allows you to design at a slightly higher level than the code itself. When you use programming-language constructs, you sink to a lower level, eliminating the main benefit of design at a higher level, and you saddle yourself with unnecessary syntactic restrictions.
- Write pseudocode at the level of intent. Describe the meaning of the approach rather than how the approach will be implemented in the target language.
- Write pseudocode at a low enough level that generating code from it will be nearly automatic. If the pseudocode is at too high a level, it can gloss over problematic details in the code. Refine the pseudocode in more and more detail until it seems as if it would be easier to simply write the code.

Cross-Reference For details on commenting at the level of intent, see “Kinds of Comments” in Section 32.4.

Once the pseudocode is written, you build the code around it and the pseudocode turns into programming-language comments. This eliminates most commenting effort. If the pseudocode follows the guidelines, the comments will be complete and meaningful.

Here’s an example of a design in pseudocode that violates virtually all the principles just described:



Example of Bad Pseudocode

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

What is the intent of this block of pseudocode? Because it’s poorly written, it’s hard to tell. This so-called pseudocode is bad because it includes target language coding details, such as **hRsrcPtr* (in specific C-language pointer notation) and *malloc()* (a spe-