**Chapter 7: High Quality Routines**

**What is a “routine”?**

* A routine is an individual method or procedure invokable for single purpose.

**Things to keep in mind**

* Descriptive name
* Is documented
* Good layouts with logical organization
* Input variable cannot be changed
* No reading and writing to global variables
* Routine should have single purpose
* Routine needs to defend itself from bad data
* Numbers need to be in variables
* All parameters must be used
* No more than 7 parameters
* Parameters should be ordered thoughtfully

**Valid Reasons to Create a Routine**

* Reduce complexity
  + create a routine to hide information so you wont have to think about it
  + pull blocks out of nested loops or conditionals
  + moving a section of code into its own routine aids readability
* Avoid duplicate code
* Support subclassing
  + You need less new code to override a short, well factored routine than a long poorly factored one
* Hide sequences
  + Forces a sequence to happen in correct order
* Hide pointer operations
  + Difficult to look at
* Improve portability
* Simplify complicated boolean tests
  + details of test are out of way
  + descriptive function name summarizes purpose
* Improve performance
  + can optimize code in one place instead of several

**Operations That Seem Too Simple to Put Into Routines**

* **A one liner with like 4 mathematical operations is much more readable if wrapped in a function**

**Design at the Routine Level**

Big focus on cohesion

High cohesion = Cosine()

Lower = CosineAndTan() b/c doing more than one thing

**The goal is to have each routine do one thing well and not do anything else.**

**Functional Cohesion**

* Strongest and best kind
* When a routine performs one and only one operation

**Less good types:**

* Sequential Cohesion
  + When a routine contains operations that must be performed in a specific order, that share data from step to step, and that don’t make up a complete function when done together
* Communicational cohesion
  + Operations in a routine make use of the same data and aren’t related in any other way
* Temporal cohesion
  + When operations are combined into a routine because they are all done at the same time
    - (Startup(), NewEmployee(), Shutdown())

**Unacceptable types:**

* Procedural cohesion
  + When operations in a routine are done in a specific order
    - (like reading in user input orderly?)
* Logical cohesion
  + When several operations are stuffed into the same routine and one of the operations is selected by a control flag that’s passed in.
* Coincidental cohesion
  + When operations in a routine have no discernable relationship to each other

**Good Routine Names**

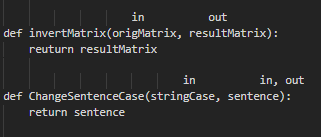
* Describe everything the routine does, and side effects
  + If the name sounds silly, change the routine functionality
* Avoid meaningless, vague or wishy-washy terms
  + HandleCalculation(), PerformServices(), OutputUser(), ProcessInput()
* To name procedure, use strong verb, followed by an object

**How Long Can a Routine Be?**

* Theoretical best max length is one screen, 50-150 lines

**How to Use Routine Parameters**

* Interfaces between routines are some of the most error prone areas of a program
* Put parameters in input-modify-output order
  + Instead of ordering parameters randomly or alphabetically, list
    - parameters that are input only first
    - input-and-output second
    - output only third



* Use all parameters
  + if passed to a routine, use it
* Put status or error variables last
* Don’t use routine parameters as working variables
  + don’t change/increment input vars, copy and assign to local
* Document interface assumptions about parameters
  + if you assume data has a certain characteristic, make note of it
    - Whether parameters are input-only, modified or output only
    - Units of numeric parameters (feet, in, meter)
    - Meanings of status codes and error vals
    - Ranges of expected values
    - Specific values that should never appear
* Limit number of parameters to about 7
* Pass the variables or objects that the routine needs to maintain its interface abstraction
* Use named parameters

**Special Considerations in the Use of Functions**

**When to use a function and when to use a procedure?**

* Purists argue that a function should return only one value
* Procedure return multiple parameters

**Setting the Functions Return Value**

* Check all possible return paths
* Don’t return references or pointers to local data