**Chapter 19: General Control Issues**

**Boolean Expressions**

* Except for the simplest of control structures, the one that calls for the execution of statements in a sequence, all control structures depend on the evaluation of Boolean expressions

Using true and false for Boolean Tests

* Use identifiers like “true” and “false” instead of 0 or 1
  + Problem with 0 and 1 is its not clear what is true and what is false
  + Sometimes its not even clear if they represent true or false at all
* Use true and false constants to make the intent clearer
  + If you don’t have to remember what 1 and 0 mean your life will be easier

Simplifying Complex Expressions

* Break complicated tests into partial tests with new Boolean variables
  + Assign intermediate results
* Move complicated expressions into Boolean functions
* Even if you only use the test once, a well named function improves readability tenfold
* Use decision tables to replace complicated conditions

Forming Boolean Expressions Positively

* “I aint not no undummy” – Homer Simpson
* Most people have a hard time understanding a lot of negatives
* Abstract Boolean test to positive sounding routine
* Flip ifs and elses if needed

Using Parentheses to Clarify Boolean

* Just because language doesn’t require, doesn’t mean its actually better

Write Numeric Expressions in Number-Line Order

* Min\_elements <= I and I <= max\_elements
* I < min\_elements or max\_elements < i

**Compound Statements (Blocks)**

* A “compound statement” or “block” us a collection of statements treated as a single statement for the purpose of controlling the flow of your program
* Use { } but the main idea is to block out the zone of the statements that distinguishes it from everything else

**Taming Dangerously Deep Nesting**

* Nesting is one of the chief culprits of confusing code
* If you have deep nesting, can redesign the tests performed into simpler routines
* You can decrease the number of nesting levels by re-testing some of the conditions
* Can also simplify by using a break block
  + However this is uncommon so only use when entire team is familiar with it and it has been adopted by the team as an accepted coding practice
* Convert nested ifs into case statements
* Factor deeply nested code into its own routine
* Use a more object oriented approach
  + Transaction class
    - Deposit subclass
    - Withdrawal subclass
    - Transfer subclass

Summary of Techniques for Reducing Deep Nesting

* Retest part of the condition
* Convert to if-then-elses
* Convert to a case statement
* Factor deeply nested code into its own routine
* Use objects and polymorphic dispatch
* Use guard clauses to exit a routine and make the nominal path through the code clearer
* Use exceptions
* Redesign deeply nested code entirely

**A Programming Foundation: Structure Programming**

* The core of structure programming is the simple idea that
  + Programs should use one-in and one-out control constructs
  + Aka single-entry, single-exit control constructs
  + These are blocks of code that only has one place it can start and one place it can end
* Structure programs progress in an orderly, disciplined way
  + Doesn’t jump around

Three Components of Structured Programming

* **Sequence**
  + Set of statements executed in order
* **Selection**
  + Control structure that decides what statements to be executed
  + If-then-else, switch, case
* **Iterations**
  + Group of statements to be executed multiple times

**Control Structures and Complexity**

* One reason so much attention has been paid to control structures is they are a big contributor to overall complexity
  + Good use decreases complexity
  + Poor use increases it
* Complexity of program is related to the amount of effort required to understand it

How Important is Complexity?

* Always take steps to reduce complexity wherever possible
* Control flow complexity is important because correlated with
  + Low reliability
  + Frequent errors

General Guidelines for Reducing Complexity

* First, can improve own mental abilities (lmaooooo)
* Second, decrease complexity of programs
  + Specifically, decrease the amount of concentration required to understand them
* How to Measure Complexity
  + Trust your intuition because hard to quantify
  + Count the number of “decision points” in a routine
    - If, and, or type statements
* What to Do with Your Complexity Measurement
  + 0-5 routine is probably fine
  + 6-10 start to think about ways to simplify the routine
  + 10+ Break part of routine into second routine and call from first routine
    - Moving part of a routine into another routine doesn’t reduce overall complexity of the program, it just moves the decision points around
    - BUT it does reduce the amount of complexity you have to deal with at any one time