Mathematics 6833 (Computing for Pure Mathematicians) – Course Outline

I. Introduction
  1. My computing autobiography
  2. Case studies: computing in research
     i. Discovering a pattern
     ii. Investigating a conjecture
     iii. How *not* to use programming in research

II. Fundamental concepts
  1. Are computers fast?
     i. The case for yes
        a. 3 gigahertz / 100 years = 1 hertz / 1 second
     ii. The case for no
        a. What the processor does
        b. Memory management: the stack and the heap
  2. Types of computing languages
     i. Low-level and high-level languages
     ii. Compiled and interpreted languages
     iii. (Very) basic data structures
     iv. Static and dynamic typing
     v. Imperative languages and structured programming
     vi. Object-oriented languages and functional languages

III. Design — The fundamental imperative is to manage complexity
  1. Encapsulation: different parts of the program are as independent as possible from each other.
  2. "Information hiding": one part of the program cannot know *how another part works*— it must know only *what the other part does* (i. e. its *interface*).
  3. Stratification: portions of the program are set up at uniform levels of abstraction
  4. Leanness: only necessary parts are present (Voltaire: "A book is finished when nothing can be added and nothing can be taken away.")
  5. Specialization: different parts of the program should be responsible for *well-defined, specific* tasks.
  6. High fan-out: many parts use a single part
     i. Good: when the used part is simple and does something specific (duplication of code to accomplish that task has been eliminated)
     ii. Bad: if the used part is a complex, overly large part that should be divided into simpler pieces with more specific roles.
  7. Reusability: setting up the project so that its parts can be used in other projects.
  8. Extensibility: setting up your project functionality can be added without having to change much of what is already there.

IV. Software correctness and testing
   1. "Test first" development
   2. Regression testing
V. The GAP language
   1. Basic syntax
   2. Example from linear algebra: matrix calculations
   3. Example from number theory: continued fraction calculator
   4. Example from group theory: testing a conjecture about generating pairs
   5. Example from knot theory: converting between the Scharlemann-Thompson invariant and the principal slope invariant
   6. Example from knot theory: calculation of the homology of cyclic branched covers
VI. Object-oriented programming
   1. Classes, objects, and inheritance
   2. The Java language
   3. The API libraries
   4. Example: a Java polynomial calculator
VII. Good coding technique
   1. Coding is writing
   2. Code layout
   3. Variables
      i. Scope
      ii. The art of naming
   4. Conditionals
   5. Iteration
   6. Commenting
   7. Refactoring
VIII. Haskell
   1. Using the HUGS interpreter
   2. Strong typing
   3. Thinking at the functional level
   4. Basic list manipulation
   5. Defining functions in Haskell
      i. Pattern recognition
      ii. The `where` keyword
      iii. Guards
   6. The `map` and `filter` functions
   7. List comprehension
   8. Recursive function definition
      i. Example from number theory: continued fractions

    ii. Example from group theory: finite abelian groups

9. More on list manipulation
10. Example from linear algebra: matrix calculations
11. The powerful `foldr` command
12. Example from linear algebra: row operations and Smith normal form
13. Lazy execution and infinite lists
14. GAP and Haskell face off: calculation of homology of cyclic branched covers
15. Haskell type classes
16. The `Maybe` type
17. Example: the partial ordering class
18. Example: calculating maximal chains using mutual recursion
19. Example from topology: implementation of surfaces in Haskell

Among the major sources for the course are:

1. *Code Complete,* by Steve McConnell
2. *Haskell: The Craft of Functional Programming,* by Simon Thompson

Both of these books are highly recommended.