SYLLABUS for CS3310 (Data & File Structures)

Kaminski

REQUIRED READINGS *(info available soon)*

1) Course Pack (orange): excerpts from "File Structures . . ." [Folk, Zoellick & Riccardi]
2) Course Pack (green): Trees & Graphs chapters [Pothering & Naps "Data Structures. . . "]
   & BTrees chapter [Miller's "File Structures . . ."]
3) Course Website (assignments, code examples, readings, algorithms, links)

homepages.wmich.edu/~kaminski/3310

PRE-REQUISITE Topics & Experiences

CS1310 (Computing Foundations) or MATH1450 (Discrete Structures) (with grade C or better)
& CS1110/CS1120 sequence (with grade C or better) (CS1/CS2 using **any language,**
where the courses include:

1) a good understanding of these **concepts** :
   - **data structures**: arrays, linked lists, stacks, queues & related algorithms
   - **programming** including design/code/ test/debug; modular, structured, OOP & PP
   - a **programming language** (same language in CS1 & CS2) you're very comfortable with
   - **also**: sequential access files, recursion, simple order of complexity, internal sorting
2) actual **programming experience**, including having:
   - used arrays, linked lists, stacks or queues, recursion, sequential access file I/O
   - done at least 10(?) programs of a reasonable size & degree of difficulty
   - worked substantially as an **individual** programmer, not mainly on group projects
   - **designed** & **written** entire programs on your own, based on a pictorial/verbal/written problem description or high-level pseudo-code algorithm from instructor
   - **tested** & **debugged** these programs on your own, having made up your own test data
   - followed detailed **specs** for the program's functionality, user interface, deliverable
   - used commonly acceptable programming **style/format** and **documentation**

CLASS FORMAT & ASSIGNMENTS The class uses an active problem-solving approach, with new material presented as a development *process* rather than as a finished *product*. The course is assignment-driven, where the programming projects incorporate knowledge and techniques used in real-world applications. Student questions will be addressed in each class.

Projects are generally multi-program applications requiring OOP, multiple classes, multiple files and module interface issues. Written specifications are provided. Significant class time is spent discussing material needed to complete the project including data and file structure concepts and algorithms, software design issues, new language concepts, testing issues and the program/application structure, functionality, and interface. Advantages and disadvantages of the projects' data and file structures are covered as well as time/space considerations and alternative approaches and structures.

Other data and file structure topics are also covered which are not used in assignment – this material will also be on the exams.

For success in the course (i.e., maximum learning as well as passing with the required "C or better", since this is a prerequisite course) students should attend every class, take detailed notes, and start working on each programming project as soon as it is introduced. This allows time for questions and problems to be addressed in class, and gives motivation for learning the new concepts and approaches presented in class, in the readings, and in the code examples.

Course GOALS To provide students with the knowledge and experiences to:

- learn new data & file structure concepts and algorithms (see "TOPICS" below)
- learn when & where these concepts would be useful in real-word applications and programming contexts
- apply many of these concepts/algorithms in programming projects
- extend the ability to describe and discuss the concepts learned using correct terminology
- do projects which give further programming practice beyond CS2-level experiences, including:
  - larger, more complex programs
  - projects and concepts presented at a more abstract level
  - new programming language concepts self-taught from code examples
  - individual (vs. team) projects so each person has the full range of experiences
  - greater independence in testing and debugging
- further develop programming skills: design, language, testing, debugging, format/style
- acquire further programming support skills and concepts
- learn further problem-solving skills
- learn to follow written specs (program functionality, user interface, project deliverables)
- further develop a professional attitude towards the programming enterprise

Learning OUTCOMES Students who earn a "C" or better in this course should be able to:

- communicate knowledgeably about data & file structures and their algorithms
- select appropriate data & file structures for a particular problem
- develop and test multiple programming applications
- use selected data & file structures in programming projects
- write readable, maintainable programs
- follow written specifications for program functionality, structure and user interface

TOPICS (not covered in this sequence)

- data structures: TREES (binary search trees, height-balanced BST, heaps)
  GRAPHS (traversals, spanning trees, minimum cost path), HASH TABLES
- file handling issues: FILE STRUCTURES (direct address, hash, indexed), RANDOM ACCESS, BINARY files, file DUMPS, BTREE indexes, relevant language issues for these concepts
- related issues for above concepts: implementation, advantages/disadvantages, practical uses, alternatives, time & space concerns
- SOFTWARE ENGINEERING issues at the "programming" level
  - program/application design, structure, format/style/readability, maintenance
  - program implementation, incremental development, new language concepts
  - program interface within an application: data files, users, designer's specs, other programs/ classes/modules, driver & utility programs, programmers
  - test plans, good test data, test drivers
  - following specifications for the program, user interface, data files, deliverable