CS 163: Data Structures

Credit Hours: 4

Course Coordinator: Karla Fant

Course Description: Data abstraction with formal specification. Elementary algorithm analysis. Basic concepts of data and its representation inside a computer. Linear, linked, and orthogonal lists; tree structures. Data structures are implemented as data abstractions. Sorting and search strategies. Data management.

Prerequisites: CS 162, Introduction to Computer Science

Goals: To acquaint students with structures used in C++ for the storage and manipulation of data. The concept of data abstraction and the problem of building implementations of abstract data types are emphasized. Both static and dynamic implementations of major structures are presented and the advantages and disadvantages of each are discussed. Structures include lists of several types, stacks, queues, trees, binary trees, B-trees and graphs. Recursion and key transformation (hashing) are examined. Students are encouraged to examine algorithms and to make judgments about the practical and social application of these algorithm concepts to large scale programming projects; the course stresses the importance of quantitative methods in designing software.

Upon the successful completion of this course students will be able to:

1. Apply data abstraction in a C++ programming problem.
2. Become proficient at evaluating the benefits and drawbacks of their design in terms of memory and run time efficiency.
3. Program using classes and linear linked lists, circular linked lists, doubly linked lists, binary search trees, arrays of linear linked lists.
4. Select the proper sorting algorithm for a problem.
5. Design solutions to problems requiring complex data structures (combinations of lists, stacks, queues, hash tables, and trees).
6. Apply data abstraction.
7. Apply both static and dynamic implementations of lists, stacks, queues, hash tables and trees.
8. Evaluate the performance tradeoffs between binary search, 2-3, 2-3-4, red-black, B-trees, and AVL trees.
9. Build and traverse data structures to manage a simple graph.
10. Apply recursion and key transformations.
11. Make judgments about the practical and social application of algorithm concepts as they apply to large scale programming projects.

Textbooks: Data Abstraction and Problem Solving with C++, Carrano, Addison Wesley.

References: Optional: Weekly Lecture Notes and Course Slides.

Major Topics: Data abstraction with axiomatic specification (4 hours)
Elementary algorithm analysis, discuss application of concepts (4 hours)
Data representation, algorithms for data structures (6 hours)
Linear lists, linked lists (4 hours)
Stacks, queues, applications (4 hours)
Tree structures (8 hours)
Sorting and search strategies (6 hours)
Data management, disk files (2 hours)
Application of social and ethical issues (2 hours)

Laboratory: Course requirements consist of homework problems, five programming problems, one midterm
Exercises:

- Design and implement an abstract data type using pointers and dynamic memory allocation (e.g., linked-list, doubly linked list, or threaded lists).
- Design, implement, and use a Stack and/or Queue abstract data type.
- Design, implement, and use a Table abstract data type with hashing.
- Design, implement, and use a tree abstract data type (e.g., binary tree, a 2-3 tree, an red-black tree, or a 2-3-4 tree).
- Design, implement, and use a quicksort and/or merge sort. Alternatively, implement a graph abstraction.

CAC Category Credits

<table>
<thead>
<tr>
<th>Category</th>
<th>Core</th>
<th>Advanced</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Structures</td>
<td>3.0</td>
<td></td>
</tr>
<tr>
<td>Algorithms</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>Software Design</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Computer Architecture</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programming Languages</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Oral and Written Communications:

Every student is required to submit at least one full page with each programming problem that discusses in paragraph form the design considerations and data structures selected. In the design considerations, they are asked to discuss what the main design considerations are, why they are the main design considerations, how they were solved, and why they were solved this way. They are encouraged to think in terms of analyzing the solution! They must answer the following questions in the design writeup:

1) How well did the data structure selected perform for the assigned application?
2) Would a different data structure work better? If so, which one and why?
3) What was efficient about your design and use of the data structure?
4) What was not efficient?
5) What would you do differently if you had more time to solve the problem?

This material is graded for grammar, spelling, style, technical content, completeness, and accuracy.

Social and Ethical Issues:

Students are encouraged to examine algorithms and to make judgements about the practical and social application of these algorithm concepts to large scale programming projects; the course stresses the importance of quantitative methods in designing software.

Theoretical Content:

Students learn in CS163 about a wide variety of linear and non-linear data structures. Emphasis is placed on learning the advantages and disadvantages of algorithms used to store, delete, search, and sort data using each of the data structures described. Primary emphasis is placed on dynamic memory management considerations and non-linear data structures.

Problem Analysis:

See Solution Design Statement.

Solution Design:

Students are expected in CS163 to use the syntactic knowledge gained in CS162 to analyze and design software using data abstraction. Students are expected to apply existing knowledge to each of the data structures discussed. Students must be able to determine the tradeoffs between selecting one data structure over another. Their designs must focus on efficiency in terms of memory usage. Optimal software performance is analyzed and stressed throughout the course.