Mapping

1		Implied. Part of every programming project.
2	**OOP**	Part of every multi-class programming project.
3	**OOP**	`Pizza` or similar.
4	**OOP**	Review – recursive linked-list, BST, heap.
5	**OOP**	`BST<T>`.
6	**Data Structures**	
7		Implied. Heapsort, Quicksort as recursive methods.
8	**Data Structures**	BST runtimes. Structure overhead.
9		Implied. Or part of **Data Structures**.
10	**OOP**	Higher-level of abstraction. Programmed.
11	**OOP**	
12	**OOP**	Included in **Theory**, too.
13	**Data Structures**	Overhead. Reading structured files.
14	**Tools**	Teach `JUnit`, `diff`, etc.
15	**Tools**	Teach `git`, `gradle`
16		

Student Learning Outcomes

`Computer Science III` deepens and broadens students *knowledge* and *skills* through hands-on development of increasingly complex computer programs. Upon finishing this course, students should be able to:

1. **OOP**. *Implement* computer programs to solve increasingly complex problems, using object-oriented programming techniques, *i.e.* **polymorphism/inheritance/interfaces**, **encapsulation**, and data **abstraction**. (Synthesize)
2. **Data Structures**. *Use* sequential, singly-linked, and **multiply-linked** data structures, along with appropriate iterative and **recursive** methods to implement efficient programmed solutions. (Synthesize)
3. **Tools**. *Use* modern software engineering tools/techniques, *i.e.* **build tools**, **version control**, **iterative development**, and **automated testing**, to improve their own programming efficacy. (Use)
4. **Theory**. *Apply* algorithm analysis and logic tools to discuss the *correctness* and *resource usage* of computer programs. (Use)

Note: It would be a *very* good idea to review this section of the syllabus before exams (especially the final). You can also use it when filling out course evaluations to review what the class taught and what it was expected to teach.

The Parts

1. *Implement* classes [**OOP**, **Data Structures**]
   a) Stand-alone classes.
   b) Implement classes against an `interface` (`implements`).
   c) Implement a *hierarchy* of classes (`extends`).
   d) Implement containers (lists, trees) of user-defined classes.
   e) Implement *generic* containers (`MyList<SomeType>`).
2. *Read/write* **structured** text files. [**Data Structures**]
3. *Trace* and *apply* recursion. [**OOP**, **Theory**]
   a) Trace recursive code like `fibonacci`.
   b) Trace A/B recursion like `isEven`/`isOdd`.
   c) Implement a fully *recursive* linked-list
   d) Implement binary search tree and heap.
   e) Implement/trace quicksort and heapsort.
4. *Apply* higher levels of abstraction to coding and data structures. [**OOP**, **Data Structures**]
   a) Identify/apply *simple* design patterns: *factory*, *decorator*.
   b) Define and apply *abstract data types*:
   Polymorphism
   Stack – including calling stack

5. **Understand** and **diagram** internal data representation [**Data Structures**, **Theory**]
   a) Describe what a **bit** and **byte** are.
   b) Describe how `char` and `char[]` are stored.
   c) Explain (and diagram) how an array, a linked-list, and a binary search tree *containing* the same type differ. Calculate the memory overhead.
   d) Explain the differences in *running time* for arrays, linked-lists, and BST containing the same data.
6. *Document* every computer program turned in. [**OOP**, **Tools**]
   a) In-line documentation: header comments, in-line comments.
   b) `README`: **Testing Plan**, problem statement.
7. *Use* `git`, `gradle`, `JUnit`. [**Tools**]

Computer Science Program Student Learning Objectives

The Computer Science department has adopted five **program** learning objectives for students completing any undergraduate major in the department [See the Departmental website for the complete list.]

`Computer Science III` addresses PSLO #2 and #3.

Students graduating from the Computer Science department at SUNY Potsdam are expected to be able to

> 2. *Solve problems through analysis and implementation of tested programs that use data structures and algorithms.*
>
> **OOP** and **Data Structures** are focused on problem solving through program implementation. **Tools** is about *supporting* the programmer in doing this.
>
> 3. *Program the multiple layers (e.g., compiler, operating system, network, assembly language) between a high-level programming language and the underlying hardware.*
>
> **OOP** is all about layers of abstraction and `Theory` includes bit-level logic and encoding.