When students have completed CS61B, Data Structures, they will:

1) Have learned the basic principles and mechanisms of object-based and object-oriented programming, as evidenced by an ability to interpret, design, and use simple classes that employ these principles and mechanisms.

2) Be able to produce encapsulated modules and abstract data types, and will understand data hiding and its importance in building large programs. They will understand how to design interfaces that partition a large project into independently implementable modules.

3) Have learned the syntax and semantics of at least one particular object-oriented language (currently Java), as evidenced by an ability to read and write simple programs in this language.

4) Understand how to analyze the complexity of algorithms, and to express it using asymptotic notation.

5) Have learned the basic concepts and techniques of testing, including unit testing, blackbox testing, whitebox testing, and regression testing, as demonstrated by an ability to write such tests for programs they develop and to arrange for their automated execution.

6) Be able to find and correct bugs in their programs, and to use a debugger to do so.

7) Have learned to use programming tools for compilation control, editing, version control, and debugging.

8) Have learned the definitions, uses, performance characteristics, and typical implementations of the following standard data types:
   a) Arrays
   b) Other sequence types: linked lists, stacks, queues
   c) Trees, including expression trees and game trees
   d) Search trees, including simple binary search trees, and balanced trees (such as 2-4 trees and splay trees)
   e) Priority queues and heaps
   f) Hash tables
   g) Pseudo-random number generators
   h) Graph structures

9) Have learned the uses, performance characteristics, and typical implementations of the following classical algorithms:
   a) Sorting and selection: insertion sort, quicksort, heap sort, merge sort, radix sort
   b) Depth-first tree and graph traversals
   c) Breadth-first tree and graph traversals
   d) Some simple, illustrative graph algorithm, such as for shortest paths
   e) Shuffling and random selection

10) Understand variable allocation, memory management, garbage collection, and their performance implications in modern programming languages.
11) Have demonstrated the ability to implement substantial programs totaling at least 1500 lines in the form of at least three programming projects, including documentation and testing.