Building Lists

Many data structures have multiple possible implementations that optimize either the speed, memory storage, or some combination therein. List share common functionality across their numerous possible implementations, the Array List or Linked List (which we can build using loops or recursion). While the exact functions and names for the functions vary slightly across languages, the following are common functions within Lists.

- size – returns the size of the list
- isEmpty – return true if the list is empty
- add – Add an item to the end of a list (returns true if added)
- insert – Add an item before the end of a list (returns true if added)
- remove by index – removes the value at the specified index (returning that item)
- clear – clears all values from the list
- get – returns the value at the specified index (or a blank string for invalid indices, for our example)
- set – replace the value at the specified index (returns the replaced value or an empty string for invalid indices)
- contains – returns true if the list includes the provided value
- indexOf – returns the index of the first occurrence of a value in the list
- lastIndexOf – returns the index of the last occurrence of a value in the list
- remove a value – removes the first occurrence of a value in the list (returns true if it removed a value)

The starter project provides each of the above features as a function you must implement and provides test cases covering each of these and a few edge cases and other such features. The project includes three options for practicing building lists. You do not have to complete all three options, nor do you have to complete each of the functions above on any one of the Lists. Your task is to practice with any or all these lists and earn the points required/desired within your course plan.

Remember – You MUST complete these entirely by hand. You cannot use any existing data structures within the language to solve these problems.

Option 1: Array List ***(~65 points)***

The array list is the simplest data structure that is quickest for many functions but slow in others. The array list uses a basic array for storage but wraps that array within logic to behave like a list. Use the Data Structures visualizer to engage with Array Lists by hand before tackling the code to realize one. The array list does not appear in the book and may not be as easy to find as an algorithm online. You can use DSVisualizer or reach out with questions for aid.

Option 2: Linked List ***(~65 points)***

The linked list is a very popular data structure that interviewers may use to test your programming knowledge. Learning the workings of a linked list not only makes it more likely you can succeed in interviews but offers clever strategies and expands your mental prowess in algorithms and visualizing data structures. The book and many online resources describe the fundamental algorithms required for linked lists. The starter Linked List code, though, should be implemented iterative (i.e., with loops). Save your recursive algorithm for option 3.

Option 3: Recursive Linked List ***(~75 points)***

Recursion offers many elegant and powerful algorithms for solving problems, especially in data structures like linked lists. In this option, you CANNOT use loops of any kind. Not every function requires recursion, but you must use recursion instead if you find you want to use a loop.