Intro to Pointer-based structures

Requirements:
- Size changes dynamically
- Items don't "move around" in memory
public class ListItem {
    int value;
    ListItem next;
    public ListItem(int v, ListItem next) {
        value = v;
        this.next = next;
    }
    public String toString() {
        if (next == null)
            return "" + value;
        else
            return value + " " + next.toString();
    }
}
List\Item a, b, c;

\begin{itemize}
  \item a = new List\Item(5, null);
  \item b = new List\Item(7, a);
  \item c = new List\Item(9, a);
  \item \textcolor{red}{c.next = b};
  \item \textcolor{red}{b.next = c.next};
\end{itemize}

Heap

Stack
List Item \( a, b, c \);

\( a = \text{new List Item}(5, \text{null}); \)

\( b = \text{new List Item}(7, a); \)

\( c = \text{new List Item}(9, a); \)

\( c.\text{next} = b; \)

\( b.\text{next} = c.\text{next}; \)

![Heap diagram]
Abstract Data Type

1. What are the methods?
   - `add(int x)`
   - `remove(int x)`
   - `contains(int x)`
   - `iterator()` (Iterator)
   - `toString()` (String)
   - `size()` (int)

Suggests:
- Keep a counter "size" of # of items
- Reference to the first item
Representation invariant -

a property of the internal representation of an object that is always true

\[ \text{size} \equiv \# \text{ of items in the list} \]

Obligation:
- Invariant must be true after the constructor runs.
- Each method "preserves" the invariant.