1) Data structures (containers, collections) help programmers organize data in programs effectively. A programmer using an “off-the-shelf” data structure (e.g., C++ STL container classes) is concerned about:
- selecting the appropriate data structure to aid in solving their programming problem,
- **how to use** the selected data structure through its API (Application Programming Interface (API)).

a) What are the concerns of the implementor of the data structure?

2) What are some examples of every day uses of a “list”?

3) What list operations would be good to include in its API?

4) The items in a list can be stored in an array (or vector) or as dynamically allocated nodes in a linked list.

<table>
<thead>
<tr>
<th>Abstract list: apples, bananas, cereal, bread, milk</th>
</tr>
</thead>
<tbody>
<tr>
<td>array implementation:</td>
</tr>
<tr>
<td>0 apples 1 bananas 2 cereal 3 bread 4 milk 5</td>
</tr>
<tr>
<td>singly-linked list implementation:</td>
</tr>
<tr>
<td>apples —— bananas —— cereal —— bread —— milk</td>
</tr>
</tbody>
</table>

a) What advantages does an array implementation have over a linked list implementation?

b) What advantages does a linked list implementation have over an array implementation?
5) Consider the book’s LinkedList Template Version 2

```cpp
// A class template for holding a SORTED linked list. 
// The node type is also a class template. 
#ifndef LINKEDLIST_H
#define LINKEDLIST_H

//*********************************************
// The ListNode class creates a type used to * 
// store a node of the linked list.           *
//*********************************************

template <class T>
class ListNode
{
public:
    T value;       // Node value
    ListNode<T> *next; // Pointer to the next node

    // Constructor
    ListNode (T nodeValue)
    { value = nodeValue;
      next = NULL; }
};

//*********************************************
// LinkedList class                           *
//*********************************************

template <class T>
class LinkedList
{
private:
    ListNode<T> *head; // List head pointer

public:
    // Constructor
    LinkedList()
    { head = NULL; }

    // Destructor
    ~LinkedList();

    // Linked list operations
    void appendNode(T newValue); // Append the newValue to the end of the list
    void insertNode(T newValue); // Inserts the newValue in sorted order
    void deleteNode(T selectValue); // Delete the selectValue
    void displayList() const; // Displays all list items to cout
};

a) Do you see anything wrong with this combination of operations?