Overview: In this lab, you will work in pairs, using test-driven development to write in C++ a scaled-down version of the LinkedList class from Java. Your linked list class must be templated so that it will work with any data type.

Specifications:

- Your class must be named LinkedList.
- You must use the STL data structure list as the private data in your LinkedList class. This will require you to use iterator and/or const_iterator associated with list. (See the C++ API for list.) Except for the return type for the toArray() method (see below), you are not allowed to use vector.
- You must implement a default constructor, copy constructor, and destructor.
- With T as the templated type of your linked list, your class must implement the following public methods:
  
  void add( T element );
  int size() const;
  T get( int index ) const;
  T remove( int index );
  vector<T> toArray() const;
  LinkedList<T>& operator+=( const T& item );

  The get method must return the appropriate T-type data item at the corresponding position, but leave the linked list unmodified. If there is no element at the corresponding position, throw an std::invalid_argument exception. (You can catch const exception& on the user side, along with using the what() method from the exception class.) Reasonable exception messages might look like the examples shown below (consider using stringstream when appropriate).

  invalid attempt to retrieve from empty list
  invalid index: -1  list size: 8
  invalid index: 100  list size: 8

- The remove method must return the appropriate T-type data item at the corresponding position, and remove the corresponding node from the linked list. If there is no element at the corresponding position, throw an exception. Use std::invalid_argument for your throw. Reasonable exception messages might look similar to those shown above.

- The toArray method will need to construct and return a C++ vector of T-type data items appropriately copied from the linked list. The linked list should remain unmodified.

- The operator+= method should append to the list, similar to the add method.

Testing: You must also submit a separate piece of source code that exhibits careful testing of your linked list implementation. (See last lab for starter suggestions.) Your tester code should, for each test, include an appropriate print statement(s) indicating what is being tested followed by output indicating correctness of the implementation.

Notes Of Interest:

- Be sure to test your implementation carefully. Does the implementation work correctly if trying to remove from an empty list? When the head is removed? When the tail is removed? When the list consists of only 1 item? Etc.

- The operator+= method will need to return a reference to the current object so that the user may do something like my_list += i; and have my_list appropriately updated.

Submitting: Package your LinkedList.h class and your tester file into a tarball named cmosc240_1ab11_netid1_netid2.tgz and drop into the lab11/ folder in the shared Box folder of one of the persons in your pair.