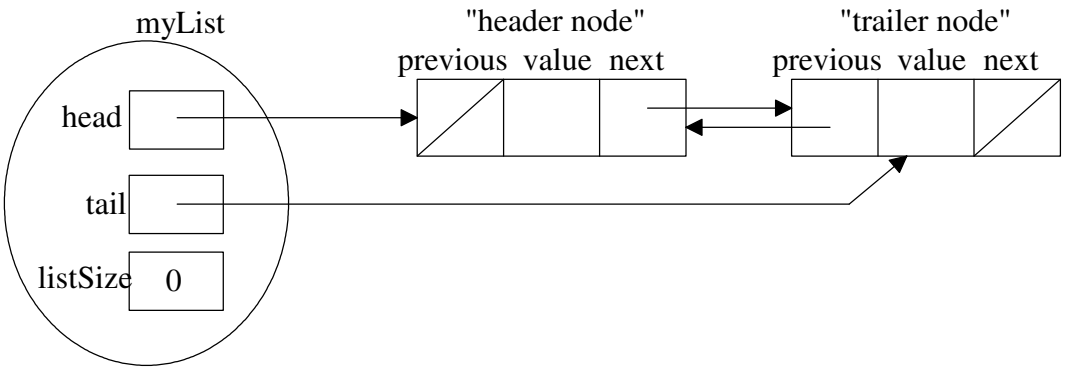
Data Structures (810:052) Lab 5 Name:_____

Objectives: You will gain experience:

- implementing a doubly-linked list data structure
- implementing an iterator for a doubly-linked list data structure

Download the following file to your desktop: http://www.cs.uni.edu/~fienup/cs052s10/labs/lab5.zip
Extract this file to the Desktop by right-clicking on lab5.zip icon and selecting Extract All.

Part A: In lecture 9 we discussed implementing a **sorted list** using a doubly-linked list with a header and trailer nodes to reduce the number of "special cases." Thus, the empty list would look like:



We even implemented the insertNode member function to insert a newValue into the sorted list.

```
template <class T>
void DoublyLinkedList<T>::insertNode(T newValue) {
      ListNode<T> *current;             // To traverse the list

      // Position current at the head of list.
      current = head->next;

      // Skip all nodes whose value is less than newValue.
      while (current != tail && current->value < newValue) {
            current = current->next;
      } // end while

      // Allocate a new node and store newValue there.
      ListNode<T> *newNode = new ListNode<T>(newValue);

      // Link newNode into doubly-linked list
      newNode->next = current;
      newNode->previous = current->previous;
      current->previous = newNode;
      newNode->previous->next = newNode;
      listSize++;
}
```

The `lab5.zip` file you downloaded and extracted contains a `SortedDoublyLinkedList` folder with a Visual Studio C++ project file: `SortedDoublyLinkedList.sln` inside. Double-click on it to open this project in Visual Studio. Your task is to implement the deleteNode member function which deletes the first occurrence of a specified value from the sorted list and returns a Boolean to indicate if it was deleted successfully.

Uncomment the code in the main.cpp file that tests the deleteNode operation.

After you have implemented and tested you deleteNode function, raise your hand and demonstrate your program.

Part B: The `DoublyLinkedList` class contains a nested `iterator` class with operators++ for walking forward through the list. The `DoublyLinkedList` class also contains `begin` and `end` functions that return iterators to the first data node and just past the last data node. The main.cpp functions makes use of these to iterator through the list.

For Part B, add to the `iterator` class operators-- for walking backward through the list, **and** add to the `DoublyLinkedList` class `rbegin` and `rend` functions that return iterators to the last data node and just before the first data node, respectively. Test your modifications from the main.cpp by walking an iterator backwards through the list. The current, partial `DoublyLinkedList` class is:

```
template <class T>
class DoublyLinkedList {
private:
	ListNode<T> *head;   // List head pointer
	ListNode<T> *tail;   // List tail pointer
	int listSize;        // count of items in the List

public:
	template <class T>
	class iterator {
	private:
		ListNode<T> * ptr;
	public:
		iterator() { ptr = NULL; }
		bool operator==(iterator<T> RHS) {
			return ptr==RHS.ptr;
		}
		bool operator!=(iterator<T> RHS) {
			return ptr != RHS.ptr;
		}
		iterator<T> & operator=(iterator<T> RHS) {
			ptr = RHS.ptr;
			return *this;
		} // end operator++

		iterator<T> & operator++() {
			if (ptr->next == NULL) {
				std::cout << "ERROR operator++:  increment past the end\n";
				exit(-1);
			} else {
				ptr = ptr->next;
			} // end if
			return *this;
		} // end operator++

		iterator<T> & operator++(int) {
			if (ptr->next == NULL) {
				std::cout << "ERROR operator++:  increment past the end\n";
				exit(-1);
			} else {
				ptr = ptr->next;
			} // end if
			return *this;
		} // end operator++
		T operator*() {
			if (ptr == NULL || ptr->next == NULL || ptr->previous == NULL) {
				std::cout << "ERROR operator*:  ITERATOR INVALID\n";
				exit(-1);
			} else {
				return ptr->value;
			} // end if
		} // end operator*
	friend class DoublyLinkedList;
	}; // end iterator class

	// Constructor
```

After implementing and testing your modifications, raise your hand and demonstrate your program.