# Learning Outcomes

Implement a linked list.

# Instructions

Our last program was to maintain a list of wizards using a vector class created by you. In this assignment, you will be writing the linked list. Implement class LinkedList consisting of Nodes of a generic type.

## Nodes

The Node struct must be represented as a template. Because this is so small, include this in your LinkedList.hpp file.

The Node struct will require these private fields.

* T data: The data to hold in the Node
* Node<T>\* next: A pointer to the next Node (if any) in the chain of Nodes.
* Node<T>\* prev: A pointer to the previous Node (if any) in the chain of Nodes.

# Before you begin.

Remove any instance to "getAssignmentCount" in any of your classes or code. It is no longer needed.

## Linked List

This class will require the following fields:

* Node<T>\* mHead: Points the first Node in the chain
* Node<T>\* mTail: Points the last Node in the chain
* int mSize: Keeps a count of the number of Nodes in the chain

Your LinkedList class must also support the following public member functions.

* LinkedList(): A default constructor sets both pointers to null and sets the size to 0.
* LinkedList(const LinkedList&): A copy constructor that takes a LinkedList object and performs a deep copy of the passed list into this list.
* ~LinkedList(): A destructor that performs pop\_front on the list while the list is not empty.
* operator=(const LinkedList&): LinkedList& An overloaded assignment operator that deletes the current array and replaces it with deep copy of the passed list
* size() const: int: Returns the size of the LinkedList.
* at(int) const: T: Returns the element stored at the specified index. If the index is out-of-bounds (either for being negative or greater than/equal the size of the list), throw invalid\\_argument("Index out of bounds.");. This will require you to include the stdexcept library. This method should operate at for accessing the first or last element in a list and at for any element other than the first or the last.
* push\_back(const T): void: Creates a new Node and assigns it to the end of the list of Nodes while updating the size by 1.
* push\_front(const T): void: Creates a new Node and assigns it to the front of the list of Nodes while updating the size by 1.
* pop\_back(): T: Deletes the Node at the end of the list (or throws an error if this is not possible). Decreases size by 1. Returns the deleted element.
* pop\_front(): T: Deletes the Node at the front of the list (or throws an error if this is not possible). Decreases size by 1. Returns the deleted element.

## Return to the Fellowship object.

Return to your Fellowship code. Find Fellowship.h and remove the import statements to "vector.hpp" and replace those with the local version of "LinkedList.hpp" that you made.

* Add a menu option to add to the beginning of the list.
* Add a menu option to remove the first wizard in the list.
* Add a menu option to remove the last wizard in the list.
* Add a menu option to remove every wizard in the list.

Test your linked list application. Add a wizard to the list 1,000 times. Add a second wizard to the list 1,000 times. Remove the first wizard in the list. Remove the last wizard in the list. Verify that you have 1,998 wizards in the list.

After you verify that there are 1,998 wizards in the list, clear the list using your newest menu option. Verify that you have 0 wizards in the list.

# What to Hand In

Upload a zip file containing the following files to the drop box on D2L:

* LinkedList.hpp - the header file containing the entire class declaration and external definitions of the LinkedList class. Normally, the definitions would be in a separate file, but class templates need to be handled differently in order for them to be compiled properly by the C++ compiler.
* All files updated from the last homework assignment that have been updated to fit this new linked list.
* Make sure your name, CSCI 3250, and Programming Assignment 3 appear in comments in all of your files.
* Note: NO CREDIT will be given to programming assignments that do not compile.
* Make sure you have compiled and tested your program before handing it in.

# My Headers

#ifndef LinkedList\_HPP  
#define LinkedList\_HPP  
#include <stdexcept>  
  
template <class T>  
struct Node {  
 T data;  
 Node<T>\* next;  
 Node<T>\* prev;  
};  
  
template <class T>  
class LinkedList {  
private:  
 Node<T>\* mHead;  
 Node<T>\* mTail;  
 int mSize;  
 void clear();  
 void copy(const LinkedList& list);  
 void checkForEmptyList();  
  
 // This is used to traverse  
 // a list for the `get` method  
 // below.  
 Node<T>\* traverse(int) const;  
  
public:  
 LinkedList();  
 LinkedList(const LinkedList&);  
 ~LinkedList();  
 LinkedList<T>& operator=(const LinkedList&);  
 int size() const;  
 T get(int) const;  
 void push\_front(T);  
 void push\_back(T);  
 T pop\_front();  
 T pop\_back();  
};

You have to write the rest.