1 Assignment

1.1 Intro

In this assignment you will create a container class that implements a linked list class that is templated so that it can contain any data type you want.

You will also need to download and install openFramework and build any one of the test applications.

1.2 Provided Files

Fork the starting git repository using the URL below. This time we are providing you `catch.hpp` and and a simply directory structure. You should develop the the `LinkedList` class in the source directory and then write tests in the test directory to verify that the code works. We will later be providing the an openFrameworks application that will work with your linked list implementation.

https://classroom.github.com/a/inhETWD0

We are providing you with two source files.

- `ll.h` - You may not change the public interface of this class. You may add constructors to the `iterator` class and the `const_iterator` though you may not add anything else to the public interface of these classes.
- `ll.hpp` - You should put all the function implementations in this file.

You may add any files you need to test or develop the class but all code to implement the `LinkedList` class must be contained in the two provided files.

2 Linked List

A linked list is a very standard data structure in computer science. It is designed to efficiently use use space when compared to an array. In a linked list you can remove an element from the list without having to copy other data as you would need to do in an array.

This data structure is implemented with a head pointer that points to the first element in the list with each element pointing to the next and finally when all items are done pointing to null. This can be seen in Figure 1.

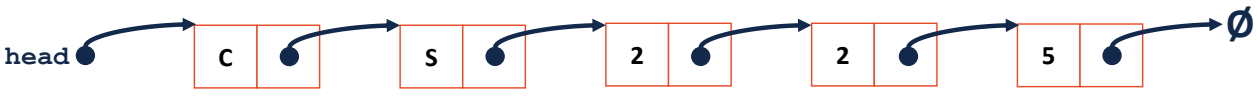

Figure 1. Linked List

The most basic operation in a linked list is to insert a node at the front of the list. This is done using the following steps.

- Create a new list node containing the item to be inserted.
- Update the next pointer in the new list node to point at the same node as head pointed to.
- Change the head node to point to the new node.

This process can be repeated several times to insert more nodes in a list.

The process to remove a node from the front of the list is similar and proceeds as follows.

- If the list is empty do nothing and stop.
- Create a temp pointer to point that points to the current head node.
- Update the head pointer to point to the next node.
- Delete the temp node.

Again to empty the list this can be repeated until the head pointer points to null. This will empty the list and deallocate all the memory.

To remove a node from later in this list you simply find the node before the node you need to remove from the list and then execute the procedure for removing as if the next pointer in the node you have found was the head pointer. Similarly to insert a node at the end of the list you need to find the last node in the list and treat that node as the head node and execute the procedure for inserting at the head with the pointer in that node being treated as the head pointer.

The following website will let you experiment with calling functions on a linked list to see how the code should behave.

`https://visualgo.net/en/list`

For more information checkout the Wikipedia link below.

`https://en.wikipedia.org/wiki/Linked_list`

3 Requirements

To complete this assignment you must decide what private data members and helper classes or functions you need to implement the following public functions for the `LinkedList` class. The linked list class public functions are the following.

- `LinkedList()` - This is the default constructor which should construct an empty linked list.
- `explicit LinkedList(const std::vector<ElementType> &values)` - This constructor will create a linked list containing in order the elements from the value vector.
- `LinkedList(const LinkedList& source)` - This function will create a new linked list that is a deep copy of the `source`.
- `LinkedList(LinkedList&& source) noexcept` - This function will implement a move constructor that will make a new linked list using the already allocated elements from the `source`.
- `~LinkedList()` - This destructor function will delete all the allocated data in the linked list class.
- `LinkedList& operator=(const LinkedList& source)` - This function is the copy assignment operator which should make a deep copy from the source deallocating all data from the old location. Remember to handle the case where the both sides are the same list.

- `LinkedList& operator=(LinkedList&& source) noexcept` - This function will copy the data from the source reusing the allocated data from the source and deleting any data that was allocated. Remember to handle the case where the both sides are the same list.
- `void push_front(ElementType value)` - This function will add a new element to the linked list at the front of the list.
- `void push_back(ElementType value)` - This function will add a new element to the linked list at the back of the list.
- `void pop_front()` - This will remove the front element from the linked list and delete the allocated data. If the list is empty it will do nothing.
- `void pop_back()` - This will remove the back element from the linked list and delete the allocated data. If the list is empty it will do nothing.
- `void RemoveOdd()` - Remove all the odd elements from the list. Remember that the list is zero indexed so the first element in the list is even. This should be implemented without directly calling any functions in this class.
- `clear()` - Delete all data in the linked list returning the list to the same state as the default constructor.
- `ElementType front() const` - Return a copy of the `ElementType` element stored in the first node of the list. This does not remove any items from the list. If the list is empty you should crash to an assert `ElementType`.
- `ElementType back() const` - Return a copy of the `ElementType` element stored in the last node of the list. This does not remove any items from the list. If the list is empty you should crash to an assert `ElementType`.
- `int size() const` - Return the number of elements in the list.
- `bool empty() const` - This function returns `true` if the list is empty otherwise returns `false`.
- `bool operator==(const LinkedList& rhs) const` - This function will compare the list element by element returning `true` if they are all equal otherwise it will return `false`.
- `iterator begin()` and `const_iterator begin() const` - These functions return an iterator and a const iterator respectively that points to the start of the list.
- `iterator end()` and `const_iterator end() const` - These functions return an iterator and a const iterator respectively that points to the past the end of the list.

You must implement some functions that work with the class outside the class as follows.

- `std::ostream& operator«(std::ostream& os, const LinkedList& list)` - This should print the `ElementType` elements stored in the list using the « operator from the `ElementType` class to print the list with a comma and a space separating each element for example "3, 2" when there are elements that would print as 3 and 2 respectively.
- `bool operator!=(const LinkedList& lhs, const LinkedList& rhs)` - This function will compare the list element by element returning `false` if they are all equal otherwise it will return `true`.

You must also implement both the const and nonconst form of the iterator interface described in

class that is `ll.h` need to make `begin()` and `end()` work.

This class must be implemented as a linked list and can not use any `STL` classes other than to handle the `explicit LinkedList(const std::vector<ElementType>& values)`. You must handle all memory yourself using `new` and `delete`.

You must implement and test all these functions. The implementations will need to be in the `ll.hpp` file.

Hints: We would like to suggest that you implement the insertion and comparison operators to test before handling the operators that remove nodes from the list since they have a higher change of having bugs. Again work in the smallest testable steps possible. Each function should be very small but some can be tricky so test each as thoroughly as possible before moving on. Finally remember that as you implement each function you will need to consider testing the earlier implemented functions with the more complex states that you may have at that time.

4 openFrameworks

You can find the openFrameworks code at the following url.

`https://openframeworks.cc/download/`

You should install using the appropriate instructions for your operating system and than build at least one of the test applications.