Programming Assignment 1 Part 1

Doubly Linked Lists

Objectives

- (Part 1) Implement a **doubly linked list** and a templated version with the provided **Abstract Data Type (ADT)** and analyze its complexity
- (Part 2) Implement a class **Record** and **Library** and perform operations on **Record** and **Doubly Linked List**

Report & Turn In

There is no written report for this assignment. There will be an oral report over the contents of this assignment in lab the week following the due date.

`!IMPORTANT` By submitting code to Mimir and/or Canvas you acknowledge and are bound by the Aggie Honor Code:

> *On my honor, as an Aggie, I have neither given nor received unauthorized aid on this academic work*
>
> *An Aggie does not lie, cheat, or steal, or tolerate those who do.*

Your submission will be taken in place of a digital/physical signature.

Turn in your code to Mimir. You should submit the following files:

- `DLList.h` and `DLList.cpp`
- `TemplatedDLList.h` and `TemplatedDLList.cpp`
- `Record.h` and `Record.cpp` (Part 2 only)
- `Library.h` and `Library.cpp` (Part 2 only)

Please also submit a zipped or tarball version of these files to Canvas.

Provided Materials

Starter code, sample mains, and a make file are provided on Github: `https://github.tamu.edu/csce221/pa2`

Sample mains are provided for testing your code.

A sample makefile is provided for helping compile your code. The targets of note are `DLList` , `TemplatedDLList` , `Record` , and `Library` . By default all compiler warnings are turned on: `-Werror -Wall -Wextra -Wpedantic` as well as debug symbols `-g` . You can override these warnings by providing a command line value for the `WARNINGS` macro (ex: `make DLList WARNINGS=""` would disable all warnings).

If you run into large quantities of output (which is common with templates). Normally you need to solve the *first* error that the compiler finds. This is the one at the top. If you're annoyed by scrolling (or produce too much output), consider output redirection into `less` to stop printing after the terminal is filled once (ex: `make DLList 2>&1 | less`). You can exit `less` by pressing `q`

GDB may be very helpful to debugging your code in this assignment. Review the resources on Canvas and ask the TAs if you have any questions.

Preamble on Doubly Linked List

A doubly linked list is a collection of nodes where each node points to the node before it and the node after it. See Canvas (or Google) for more information.

(Part 1) The Doubly Linked List

Implement the Doubly Linked List

You will be implementing two (or one) versions of the Doubly Linked List. One version will store `int` and be called `DLList` while the other will make use of templates to store a generic type `T` and be called `TemplatedDLList`

> *Note: if you are confident in your programming abilities, you can implement only `TemplatedDLList` and then in your `DLList.h` simply include `TemplatedDLList` and have a using statement to alias `TemplatedDLList<int>` into `DLList`*

> *Note: Remember all templated items **must** be in on compilation unit, namely one `.h` file. Also remember that function bodies for non-templated items **must appear only once** in the compilation (i.e. in their own `.cpp` file)*

Each variant of *DLL* will require a helper class `DLListNode` implemented as a sub-class of the containing class. This class store a the element as well as the pointers to the next and previous nodes in the *DLL*. You can make all of `DLListNode` public, and implement any of the following that you need:

- *default constructor*
- *parameterized constructor* - This one is provided and is probably the only one you need.
- *copy constructor*
- *copy assignment*
- `operator<<`

> *Note: `DLListNode` will be implicitly templated in the case of `TemplatedDLList`. You will need initialize the stored type `T` to some value. Don't forget that you can make this someone else's problem and simply call the default constructor `T()`*

> *Node: Since `DLListNode` is a sub-class, you have to access it by scoping to the containing class first: `TemplatedDLList<T>::DLListNode`. When already inside the containing class, scoping is not required.*

For **each** of the variants of *Doubly Linked List (DLL)* implement the following member functions:

- *default constructor*
- *copy constructor* - ensure this is a deep copy
- *copy assignment operator* - ensure this is a deep copy, do not leak memory
- *move constructor*
- *move assignment operator* - do not leak memory
- *destructor* - ensure that your program has no memory leaks
- `DLListNode* void first_node(void) const` - return a pointer to the first node of the *DLL*. If the *DLL* is empty, return the same as `after_last()`

- `DLListNode* after_last_node(void) const` - return a pointer to the node after the last element of the *DLL*
- `bool is_empty(void) const`
- `T first(void) const` - return the first element in the *DLL*. Throw an exception if the *DLL* is empty.
- `T last(void) const` - return the last element in the *DLL*. Throw an exception if the *DLL* is empty.
- `void insert_first(T)` - insert the element `T` into the front of the list
- `T remove_first(void)` - remove and return the first element; throw an exception if there is no element to remove.
- `void insert_last(T)` - insert the element `T` into the end of the list
- `T remove_last(void)` - remove and return the last element; throw and exception if there is no element to remove.
- `insert_after(T, &N)` - insert the element `T` immediately after the position `N`; throw an exception if the position `N` is invalid
- `insert_before(T, &N)` - insert the element `T` immediately before the position `N`; throw an exception if the position `N` is invalid
- `T remove_after(&N)` - remove and return the element after `N`. Throw an exception if no element exists or position `N` is invalid
- `T remove_before(&N)` - remove and return the element before `N`. Throw an exception if no element exists or position `N` is invalid
- `void make_empty(void)`

Outside the class implement

- `operator<<` - The output format should have elements separated by commas and spaces: `1, 2, 3,`. Please leave the trailing comma, I don't trust my regex expression to match without it.

A word on style

The recommended implementation of *DLL* uses two additional nodes called **sentinel nodes**. At object initialization (when the constructors are called), these nodes are the *head* and *tail* of the list and point to each other. However, the list is still considered empty. Inserts are then inserted between the sentinel head and sentinel tail. This drastically simplifies the insert and remove functions by ensuring there is always something on both sides of the remove function. You are not required to use this sentinel node model of the *DLL*, however if you choose not to use them, the TAs will likely be unable to help you locate and fix and pointer problems you may create.

Take a moment and consider exactly which functions *need* to be implemented from the ground up and which ones can be implemented via others. Generally, you only *need* to implement one insert and one remove function. The other inserts and removes can be built on top of these two (as long as you use the sentinel model).

> *Example:* `insert_after` *some node N is the same as* `insert_before` *whatever node comes after N. Here is another place where sentinel nodes greatly simplify the code you need to write.*

Also, be alert to which functions take pointers and which take references. This is a common tripping point for students in this assignment.

(Part 2) Implement the class Record and Library

The class Record

You will implement a custom class record that stores the following information about a book:

- `title`
- `author` - author's name
- `ISBN` - can be a string
- `year` - can be a string
- `edition`

Define the following functions *outside* the class

- `string get_title(void) const;`
- `string get_author(void) const;`
- `string get_ISBN(void) const;`
- `string get_year(void) const;`
- `string get_edition(void) const;`

Define the following functions *outside* the class

- `operator>>` - to read a record. A sample input is given to you in `Book.txt`. Each record will contain the following information, one element per line in the following order. There may or may not be new lines between distinct records. There may or may not be trailing newlines at the input files. Records will always be complete.
  - The order: `title`, `author`, `ISBN`, `year`, `edition`
- `operator<<` - to print a record on the screen. Print the fields one per line in the order they occur in the input file.
- `operator==` - compare Records based on `title`, `author`, and `ISBN`. Return *true* if and only if all three fields match exactly between two records.
  - `bool operator==(const Record& r1, const Record& r2);`

> *Note: you probably will need to use the* `friend` *keyword for implementing operators (especially* `operator>>` *)*

> *Note:* `operator<<` *and* `operator>>` *should be each other's inverse. If you read an input with* `operator>>` *and write it out with* `operator<<`*, the resulting file should exactly match the original.*

The class Library

You should implement a library management system to store books. The system stores each book title, author's name, 13-digit ISBN, publishing year, and edition number. It is possible to have the same title and author's name for a book if there are more than one edition

To speed up the search in the library management system, the data will be stored in a vector of 26 sorted doubly linked lists. To speed up the search, each doubly linked list must be maintained in sorted order by title, author name, and ISBN (in this order). You can treat ISBN as a string. The sorting is done in alphabetical order with respect to the first letter of the book title, a letter is from A to Z.

Example:

The eighth element of the vector, i.e. the eighth doubly linked list, v[7], corresponds to the letter 'H'. I may contain, for instance, the following book records:

"H is for Hawk", Helen Macdonald, 978-0802123411, 2015, 1st edition

"Harry Potter And The Chamber Of Secrets", J. K. Rowling, 978-0439060000, 2000, 1st edition

"Harry Potter And The Chamber Of Secrets", J. K. Rowling, 978-0439060001, 2000, 2nd edition

Note: you may find it helpful to implement `operator<` *for* `Record` *and also maybe for* `DLListNode` *. This operator should go outside the class and be const qualified:* `bool operator<(const Record& lhs, const Record& rhs)`

Library Interface

Note: This section is under construction. Check back in a couple days for more details.

Your library management system should provide a friendly interface for users to create a book database and search in this database.

- The user will be asked to input the title to start searching.
- If the program does not find a book with the requested title, the user will be asked to add this title to the database, and he/she needs to provide all the required book information.
- If more than one book have the same title and author's name, these records will be displayed, and the user needs to decide which book edition to select.
- Finally, the program will display the book.