Assignment 3 Part 1 Report

May 9, 2017

Program Description This program contains the implementation of both the Doubly Linked List and a templated version of the Doubly Linked List each both accompanied by extensive test cases that test program functionality and reliability. The Doubly Linked List contains the Linked List Data Structure as well as the node Abstract Data Type in order to visualize the data of the Doubly Linked List. The testing cases are provided in each program's respective main file.

Purpose of the Assignment The purpose of this assignment was to explore and learn about the Linked List Data Structure we had talked about in lecture by implementing both the Linked List data structure and a templated version and also provide extensive testing for the program. Knowledge about generic programming and template types was also gained implementing the templated version of the Doubly Linked List.

Data Structures Description The data structure utilized in this program was the Doubly Linked List Data Structure which is essentialy a list of sequential nodes that are allocated on the heap that point to either the next or previous node and can be traversed in either direction. Each node consists of a pointer to the next and previous node in the list as well as a data value which can be accessed. Initiallity, an empty Doubly Linked List is created with two sentinel nodes that point to each other called header and trailer that are initialized on the Stack. When nodes are added to the list, the header sentinel node points to the first node in the Linked List that is allocated on the heap, and the trailer sentinel node points to the last node in the Linked List that is allocated on the heap. Overall, the Linked List data structure is implemented with several public member functions used to organize, access, and store data of different types. Nodes can be added and removed from the Linked List, the values can be read from the first and last nodes, the Linked List can be copied, assigned, and cleared, and, the Linked List can be printed via using the respective public member class functions.

Algorithm Description The main algorithms worth noting in this program are primarily associated with traversing the Linked List in order to allocate, delete, copy, assign, clear, or print a Linked List all with a complexity respective to $\mathbf{O(N)}$; otherwise, all the other functions are constant $\mathbf{O(1)}$ complexity because the functions are mainly getters and remove functions for the first and last nodes in the list which are directly accessed.

Doubly Linked List Functions

DoublyLinkedList() - default constructor, initializes a new Linked List, $\mathbf{O(1)}$

DoublyLinkedList() - copy constructor, copies each node's data value from on list to another by creating a new node for every node in the source list, $\mathbf{O(N)}$

operator=() - assignment operator, clears the current list, then assigns the data values over to the current from the source list, $\mathbf{O(N)}$

getFirst() - returns a pointer to the header sentinel node, $\mathbf{O(1)}$

getAfterLast() - returns a pointer to the trailer sentinel node, $\mathbf{O(1)}$

isEmpty() - checks if the list is empty, $\mathbf{O(1)}$

first() - returns the first node's data value if list is not empty, $\mathbf{O(1)}$

last() - returns the last node's data value if list is not empty, $\mathbf{O(1)}$

insertFirst() - inserts a new node of specified data type value in the front of the list, $\mathbf{O(1)}$

insertLast() - inserts a new node of specified data type value at the end of the list, $\mathbf{O(1)}$

removeFirst() - removes the first node in the list and returns its node value, **O(1)**
removeLast() - removes the last node in the list and returns its node value, **O(1)**
clear() - clears the list, **O(N)**
operator<<() - output operator, prints the linked list, **O(N)**

How to Compile and Run

1. In a Unix terminal, navigate to the respective source directory /**DoublyLinkedList** or /**TemplateDoublyLinkedList** using the **cd** command.
2. Once in either the /**DoublyLinkedList** or /**TemplateDoublyLinkedList** directory type **make** to compile the respective program.
3. Run the executable for the DoublyLinkedList program by typing **./run-dll** , or **./run-tdll** for the TemplateDoublyLinkedList program.

Logical Exceptions There are logical exceptions for some public member functions of the DoublyLinkedList class, primarily in the first(), last(), removeFirst(), and removeLast() functions in which the empty Linked List case is handled for the respective operations in the functions.

C++ Object Oriented Features There are two classes, one for the Linked List data structure and one for the ListNode ADT. There is also exceptions and elements of inheritance in the program which occurs in the DoublyLinkedList class when `RuntimeException` is overridden.

Tests The following are output from these programs:
Double Linked List:

```
// Construct a linked list with header & trailer
cout << "Create a new list" << endl;
DoublyLinkedList dll;
cout << "list: " << dll << endl << endl;

// Insert 10 nodes at the end with value 10,20,30,..,100
cout << "Insert 10 nodes at tail with value 10,20,30,..,100" <<
for (int i=10;i<=100;i+=10) {
  dll.insertLast(i);
}
cout << "list: " << dll << endl << endl;

// Check first and last elements
cout << "dll first: " << dll.first() << endl;
cout << "dll last: " << dll.last() << endl;
cout << endl;

// Insert 10 nodes at the beginninhg with value 10,20,30,..,100
cout << "Insert 10 nodes at front with value 10,20,30,..,100" <<
for (int i=10;i<=100;i+=10) {
  dll.insertFirst(i);
}
cout << "list: " << dll << endl << endl;

// Check first and last elements
cout << "dll first: " << dll.first() << endl;
cout << "dll last: " << dll.last() << endl;
cout << endl;

// Copy to a new list
cout << "Copy to a new list" << endl;
DoublyLinkedList dll2(dll);
cout << "list2: " << dll2 << endl << endl;

// Assign to another new list
cout << "Assign to another new list" << endl;
DoublyLinkedList dll3;
dll3=dll;
cout << "list3: " << dll3 << endl << endl;

// Delete the last 5 nodes
cout << "Delete the last 5 nodes" << endl;
for (int i=0; i < 5; i++) {
  dll.removeLast();
}
```

by the instructor on the instructor website.The testing cases cover the use of the public member functions listed above in the **Algorithm Description** Section. A new list is created and new nodes are added to the list. The testing cases then go on to check the first and last node values, copy the data to another linked list, assign data to another linked list, and delete nodes from the list. The DoublyLinkedList is then tested further with the extra testing cases provided in the main.cpp shown above to test the functionality of the isEmpty(), first(), last() and clear() public member functions.

Templated Doubly Linked List:

```cpp
// ---------------String data type testing---------------//
cout << "---------------String data type testing---------------" <
// Construct a linked list with header & trailer
cout << "Create a new list" << endl;
DoublyLinkedList<string> dll;
cout << "list: " << dll << endl << endl;

// Insert 10 nodes at back with value 10,20,30,..,100
cout << "Insert 10 nodes at back with value 10,20,30,..,100" << e
for (int i=10; i<=100; i+=10) {
  stringstream ss;
  ss << i;
  dll.insertLast(ss.str());
}
cout << "list: " << dll << endl << endl;

// Insert 10 nodes at front with value 10,20,30,..,100
cout << "Insert 10 nodes at front with value 10,20,30,..,100" <<
for (int i=10; i<=100; i+=10) {
  stringstream ss;
  ss << i;
  dll.insertFirst(ss.str());
}
cout << "list: " << dll << endl << endl;

// Copy to a new list
cout << "Copy to a new list" << endl;
DoublyLinkedList<string> dll2(dll);
cout << "list2: " << dll2 << endl << endl;

// Assign to another new list
cout << "Assign to another new list" << endl;
DoublyLinkedList<string> dll3;
dll3=dll;
cout << "list3: " << dll3 << endl << endl;

// Delete the last 10 nodes
cout << "Delete the last 10 nodes" << endl;
for (int i=0; i<10; i++) {
  dll.removeLast();
}
cout << "list: " << dll << endl << endl;

// Delete the first 10 nodes
cout << "Delete the first 10 nodes" << endl;
```

The primary testing done on the templated Doubly Linked List program was based from the testing cases provided by the instructor on the instructor website. The testing cases cover the use of the public member functions listed above in the **Algorithm Description** Section, but for this templated version, a generic data type is utilized for the ListNode data type; therefore, a Linked List can be created and can support any user-defined data type, string, double, int, char, etc. For the instructor-given testing cases, the templated Linked List is tested for the string data type to test the public member functions of the Templated Doubly Linked List class. To further test the generic-type class, extra testing cases were implemented to test the double and int data type; extra cases were also added for these data types as well, as shown above.

Conclusion Overall, extensive knowledge about the Linked List Data structure, Big-O analysis of algorithms used in the class, and generic programming principles were obtained by implementing the both Doubly Linked List and Templated Doubly Linked List.