# James Hudspeth’s Doubly Linked List Sorting

This is a doubly linked list built in C++ with two sorting functions. It is a linked data structure that consists of data stored in nodes sequentially. Each node contains two pointers, linking each node to the previous and next node. The head and tail of this list are null nodes that do not contain data and for the purposes of the program, virtually ignored. There are two different methods of sorting implemented, insertion sort and QuickSort (QuickSort does currently have a known bug where it may not sort the last number in the list). The program itself makes use of a simple user input menu. After each command the menu will reappear and ask the user what they wish to do next. The user enters commands in the form of a single digit corresponding to each menu option. The doubly linked list class is templated, allowing it to work with multiple different types of data, but the list created is currently set to int.

# Member Variables

Node

element – data stored (public)

next – pointer to next node (public)

previous – pointer to previous node (public)

Iterator

actNode – node variable that stores the current location during iteration (private)

DLList

head – pointer to the head node (private)

tail – pointer to the tail node (private)

Main

menu – char variable to hold user input for menu options

intList – doubly linked list for integers

testList – doubly linked list for testing purposes

value – user input to pass to list functions

foo – junk variable, used to end program

testFoo – testing variable for passing values to functions

fileIn – ifstream file for file import

# Functions/Methods

Node class

Node – constructor for node class

Parameters – (const T &e, Node \*n, Node \*p)

Const T &e – element to be inserted

Node \*n – pointer to next node

Node \*p – pointer to previous node

Return – is a constructor for a class

~Node – deconstructor for node class

Parameters – none

Return – deconstructor for a class

Iterator class

Iterator – constructor for iterator class

Parameters – (Node<T> \*node) node to begin the iteration from

Return – constructor for a class

~Iterator – deconstructor for iterator class

Parameters – none

Return – deconstructor for iterator class

nextNode – iterates to the next node in the list

Parameters – none

Return – actNode node

prevNode – iterates to the previous node in the list

Parameters – none

Return – actNode node

DLList class

DLList – constructor for DLList class

Parameters – none

Return – constructor for a class

~DLList – deconstructor for DLList class

Parameters – none

Return – deconstructor for a class

isEmpty – simple function to check if the list is currently empty of data.

Parameters – none

Return – true if list is emty, false if list contains data

Insert – function to insert data to the list.

Parameters – (T &e) value to be inserted into the list, defined in DoublyLinkedList as an alias of its first template parameter (T).

Return – none

Delete – function to delete data from the list.

Parameters – (Node<T> \*r) address of node to be deleted

Return – true if the value is deleted, false if not

Clear – clears the whole tree of data.

Parameters – none

Return – none

getItHead – iterates through the list from the head

Parameters – none

Return – Iterator<T> i

getItTail – iterates through the list from the tail

Parameters – none

Return – Iterator<T> i

Find – finds a value within the list

Parameters – (T &value) the value to be found within the list

Return – Node<T>\* tempNode

InsertionSort – sorts the list using the insertion method

Parameters – none

Return – none

GetHead – returns the head of the list

Parameters – none

Return – Node<T>\* head

Non-class functions

Swap – function to swap two different variables

Parameters – (T\* a, T\* b)

T\* a – first variable to swap

T\* b – second variable to swap

Return – none

LastNode – finds the last node within the list

Parameters – (DLList<T> \*root) list for the function to run on

Return – Node<T>\* last

Partition – function that runs the swap functions to sort the list

Parameters – (Node<T> \*nextNode, Node<T> \*last)

Node<T> \*nextNode – the next node within the list

Node<T> \*last – the last node within the list

Return – Node<T>\* tempNode

\_quickSort – internal function for QuickSort, iterates through the list and runs Partition function

Parameters – (DLList<T>\* list, Node<T> \*last)

DLList<T>\* list – the list for QuickSort to sort

Node<T> \*last – last node of the list

Return – none

\_quickSort – overloaded internal function for QuickSort

Parameters – (Node<T>\* nextPart, Node<T> \*last)

Node<T>\* nextPart – the next node in the list

Node<T> \*last – the last node in the list

Return – none

QuickSort – sorts a list using the QuickSort method. There is a known bug where the last few nodes may not be sorted. This function runs the \_quickSort function

Parameters – (DLList<T> \*head) a list starting from the head

Return – none

Performance Times in Nanoseconds:

|  |  |  |
| --- | --- | --- |
|  | Insertion | QuickSort |
| 10 Numbers | 1,555 | 2,372 |
| 100 Numbers | 16,869 | 21,746 |
| 1,000 Numbers | 1,329,409 | 146,740 |
| 10,000 Numbers | 89,040,923 | 2,031,397 |
| 100,000 Numbers | 9,427,753,736 | 18,659,947 |
| 1,000,000 Numbers | 1,184,362,099,657 | 213,024,956 |