Doubly-Linked List Challenge

Matthew CntKillMe

December 28, 2018

1 Introduction

Your task is to implement a doubly-linked list data structure with the given interface. You will be judged based on the following criteria (in order of most important to least important):

- Correct implementation of the interface.
- Proper memory management.
- Code readability, reuse, and performance.

The additional challenges are optional, however it is recommended for those who want more of a challenge. For the sake of simplicity, assume no boundary errors and memory allocation errors can take place.

2 Additional Challenges

2.1 Extra Functionality Interface

Make the linked list more useful by implementing practical functions.

2.2 Iterator Interface

Make iterating over a linked list more practical and efficient by implementing the iterator functions.

2.3 Extra Iterator Functionaliy Reference

Make iterators more useful by implementing some more practical functions. This challenge requires the Iterator Interface challenge to be completed.

3 Interface Reference

See *linked_list.h* for the implementation interface.
The expected runtime complexity does not account for library function calls, assume those are O(1). Variable n refers to the size of the linked_list.

Required Interface Reference		
linked_list_*	**Runtime Complexity**	**Iterator Invalidation**
init	$O(1)$	
copy	$O(n)$	
clear	$O(n)$	$[begin, end)$
resize	$O(\|n - newSize\|)$	$[begin + newSize, end)$
size	$O(1)$	
front	$O(1)$	
back	$O(1)$	
push_front	$O(1)$	
push_back	$O(1)$	
pop_front	$O(1)$	$first$
pop_back	$O(1)$	$last$
get	$O(idx)$	
set	$O(idx)$	

Extra Functionality Interface		
linked_list_*	**Runtime Complexity**	**Iterator Invalidation**
reverse	$O(n)$	$undefined$
sort	$O(n^2)$	$undefined$
append	$O(1)$	
foreach	$O(n)$	
swap	$O(1)$	

Iterator Interface Reference		
linked_list_*	**Runtime Complexity**	**Iterator Invalidation**
begin	$O(1)$	
end	$O(1)$	
read	$O(1)$	
write	$O(1)$	
advance	$O(steps)$	
insert	$O(1)$	
erase	$O(1)$	iter
dist	$O(\|dist(iter1, iter2)\|)$	

Extra Iterator Functionality Reference		
linked_list_*	**Runtime Complexity**	**Iterator Invalidation**
insert_many	$O(count)$	
insert_range	$O(\|dist(first, last)\|)$	
erase_range	$O(\|dist(first, last)\|)$	$[first, last)$
swap_nodes	$O(1)$	
reverse_nodes	$O(dist(first, last))$	
sort_nodes	$O(dist^2(first, last))$	