C Review

GWU CSCI 3411 - Fall 2019 - Lab 2
Prepared by James Taylor (Fall 2017)
Edited by Ratnadeep Bhattacharya (Fall 2019)

August 31, 2019

Topics

In this lab, you will exercise some of the most basic principles of C in preparation for your work in this course. This lab is designed to remind you of C fundamentals and operations with a primary emphasis on pointers.

1 Basic Linked List

Recall the linked list data structure allows elements to be dynamically inserted or removed from a set. A basic *linked list* consists of a reference to the `head` of the list. Each element in the list may be called a *node* and each node minimally consists of a reference to the subsequent node in the list which is typically called `next`. Each node may also reference data associated with that node. The list can be traversed by iterating through the list through `next` references.

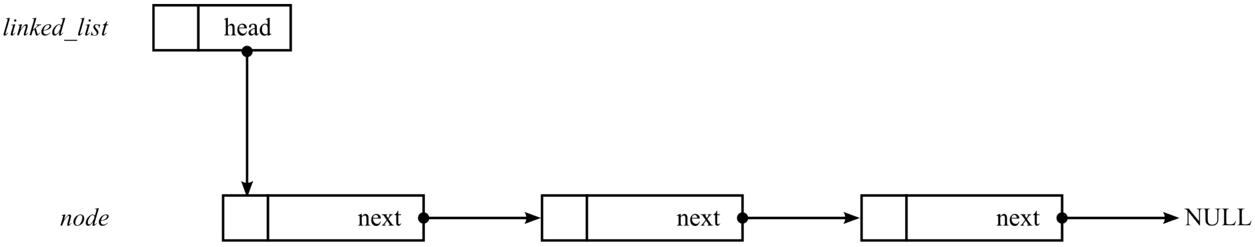

Figure 1: Basic Linked List

1.1 $O(c)$ Insertion List

The basic linked list is inefficient for insertion. In order to insert a new item into the list, the entire list must be traversed which means that in order to insert an element at the end of the list, n operations must be performed. The number of operations can be reduced by introducing a `tail` reference. The `tail` points to the last element of the list and allows an insertion to be performed in constant time by following the `next` reference on the current `tail` and then updating the `tail` to reference the item just inserted.

2 Doubly-Linked List

The basic linked list can only be traversed from front to back. A more sophisticated version of the linked list is the doubly linked list. The doubly-linked list introduces an additional pointer for each node in the list called `prev` which

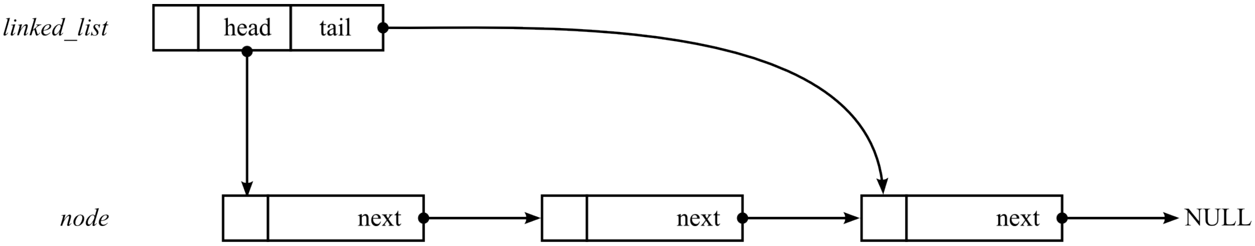

Figure 2: $O(1)$ Insertion Linked List

references the previous node. The doubly-linked list allows the list to be traversed from either the `head` or `tail` and references to all neighbors in the list can be followed or maintained from any node in the list. The drawback of the doubly-linked list is that the number of references in the list is doubled and requires more careful attention by the programmer when mainting the list.

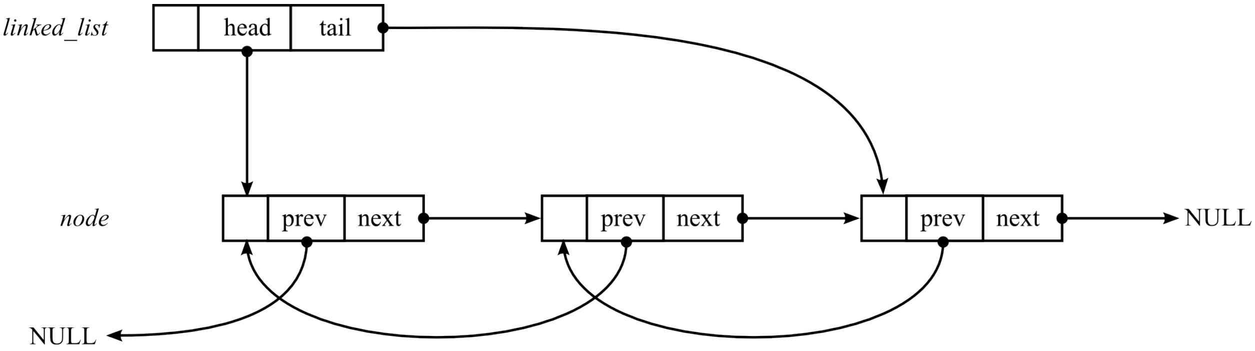

Figure 3: Doubly Linked List

3 Devices

In this lab, we are simulating a list of devices on a system. Each device will be identified by a name, a major number and a minor number. This list will be implemented as a doubly linked list. Each node of the doubly linked list will hold the device name - a char array, a major and a minor number - unsigned ints. The following methods must be implemented on the list:

1. create - to create the list. Return a newly created doubly linked list.
2. insert - insert to the end of the list. Ideally, insertion should happen by major number and then minor number - devices should be sorted by major number, all devices with the same major number should be together and within each major number devices should be sorted by minor numbers.
3. at_index - return -1 if the device is not found; the index otherwise. Two devices are considered to be the same if they have the same major and minor numbers.
4. del - delete a device from the tail of the list; alternatively build in support to delete specific devices. Return the device if successfully deleted; NULL otherwise. Note: Can you use at_index inside del?
5. size - should return the size of the list
6. is_empty - check if the list is empty

7. print - print the devices in the list

8. destroy - destroy the list. You can decide whether to delete only if list is completely empty or if to delete constituent data nodes while destroying. Return 0 on successfully destroying the list; 1 otherwise.