(More) Fun with Pointers and Linked Lists!

CS 16: Solving Problems with Computers I
Lecture #17

Ziad Matni
Dept. of Computer Science, UCSB

Administrative

- Homework situation: NO MORE HOMEWORK! ☺
- Labs: Lab10 due on Friday

FINAL IS COMING!

- Material: ***Everything*****!**
- Homework, Labs, Lectures, Textbook
- **Tuesday, 6/12** in this classroom
- **Starts at 4:00pm **SHARP****
- Duration: **3 hours long**
- ***BRING YOUR UCSB IDs PLEASE! Arrive 10-15 minutes early***
- Closed book: no calculators, no phones, no computers
- Only 1 sheet SINGLE-SIDED of written notes
  - Must be no bigger than 8.5” x 11”
  - You have to turn it in with the exam
- **You will write your answers on the exam sheet itself.**

Lecture Outline

- More exercises using pointers and linked lists

Exercise Example 1

- We've already demonstrated how to add nodes to a LL, but what about deleting them?

Figure Out the Algorithm!

Regular case:

How do I remove "B" from the LL? And get to:

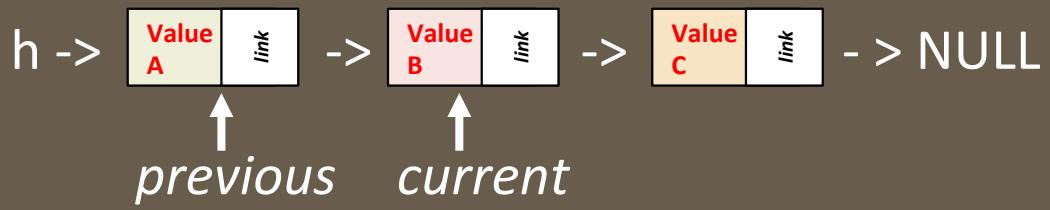
???

Algorithm for Deletion

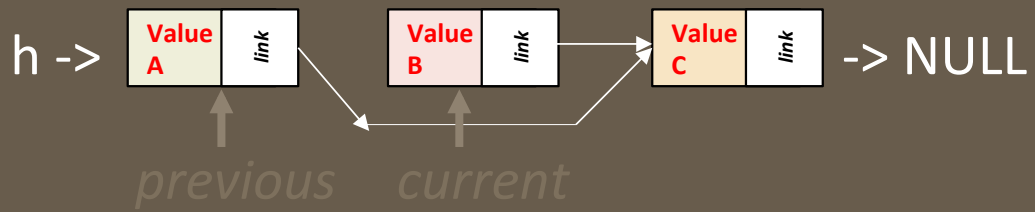
1. Find the node to delete
   a) Either by the value (or one of the values) in the node
   b) Or by its position in the linked list

2. Get a pointer to point to that node (call it *current*)
3. Get a pointer to point to the node before it (call it *previous*)

Algorithm for Deletion



4. Have *previous->link* be pointing to what's ***after*** *current*
5. Should I make *current->link* point to NULL?



What happens to Node "B"??

You have to "de-allocate" it from memory

Use: `delete(current)`

Edge Cases

- Will our algorithm work for ALL cases of a linked list?

- What about:
  1. The node to delete is a the start of the linked list?
  2. The node to delete is a the tail of the linked list?
  3. If the linked list has only ONE component?
  4. If the linked list has NO components (h -> NULL)?
  5. If I CAN'T FIND my intended node to delete?

  – Other situations???

Edge Case 1

Case of: h -> (DeleteThis) -> NodeX -> NodeY … *etc* …

- Can I just skip the first node in a simple way?
  - Yes!
- So it's a "special case"…

Edge Case 2

Case of: h -> NodeX -> NodeY -> (DeleteThis) -> NULL

- *Can I make previous = pointer to NodeY?*
- *Can I make current = pointer to "DeleteThis" node?*

- Yes and yes
- So… no "special case"…

Recall:
4. Have *previous->link* be pointing to what's ***after*** *current*

Edge Case 3

Case of: h -> (DeleteThis) -> NULL

- *Is this different from Case 1?*
- No

Edge Case 4

Case of: h -> NULL

- *Should I even try?*
  - No
- How do I check for this?
  - Hmmmm….
- "Special case"…

Edge Case 5

- What if the search criteria fails?
  - I cannot find a node at *that position*
  - I cannot find a node value equal to my target value
- Sounds like a modification to my “while loop”…
- Would the requirements for edge case 4 fit into this?
  - Yes

Entire Algorithm

1. Have head and target defined (passed into function)
2. Create 2 pointers to nodes: *current = previous = head*
3. If (head == NULL):
   a) Empty list – nothing to find
   b) Return
4. Otherwise (head != NULL):
   a) Advance thru the LL with a while loop
      i. *previous = current* and *current = current -> next*
   b) If (current = NULL), then we didn't find anything (special case: target not found)
      i. Return
   c) If (current == head), then our target is at the head (special case: skip first node)
      a) Adjust head to head->next
   d) Otherwise, it's the "regular case": previous->link = current->link
   e) Delete the node from memory! (i.e. `delete(current)`)

Entire Code Revealed

Demo Code!

```
void deleteNode(NodePtr &head, int target)
{
    NodePtr curr = head, prev = head;

    if(head == NULL)
        cout<<"Nothing to delete.\n";

    else
    {
        while
((curr != NULL) && (curr->data != target))
        {
            prev = curr;
            curr = curr->next;
        } // end while
```

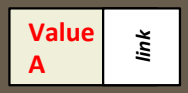
```
    // Special Case: target not found
    if(curr == NULL)
    {
        cout <<
    "Node not found – nothing to delete.\n";
        return;
    }

    // Special Case: target found at head of LL
    if(curr == head)
        head = head->next;
    // Regular case:
    else
        prev->next = curr->next;
    // Free up that now deleted node in memory!
    delete(curr);
    } // end else
} // end deleteNode
```

Exercise Example 2

- We've already demonstrated how to build a linked list using the "add to head" approach, like:

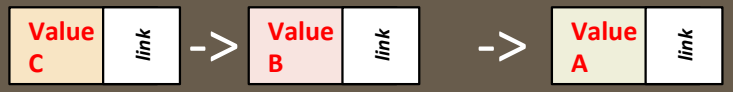
h -> NULL

h -> 

-> NULL

h -> 

-> NULL

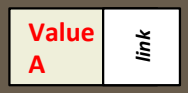
h -> 

-> NULL

Exercise Example 2

- What would it be like to build a linked list by putting new nodes at the ***tail*** instead? (without using reversing)

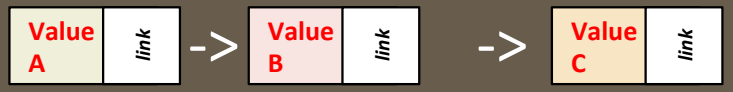
h -> NULL

h -> 

-> NULL

h -> 

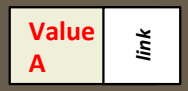
-> NULL

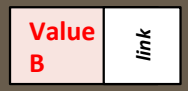
h -> 

-> NULL

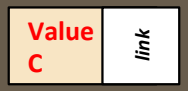
Figure Out the Algorithm!

Regular case:

h -> 

-> 

-> NULL

Here's the node… 

How do I get this in the LL?

Edge case:

h -> NULL

??? Do I do anything different here?

YOUR TO-DOs

- ❑ Lab 10 due on Friday
- ❑ NO HOMEWORK!!
- ❑ Prepare for final exam and come with questions on Thursday!
- ❑ Visit TAs‘ office hours if you need help!

</LECTURE>