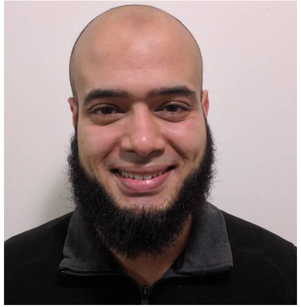
Data Structures

Nodes Deletion

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

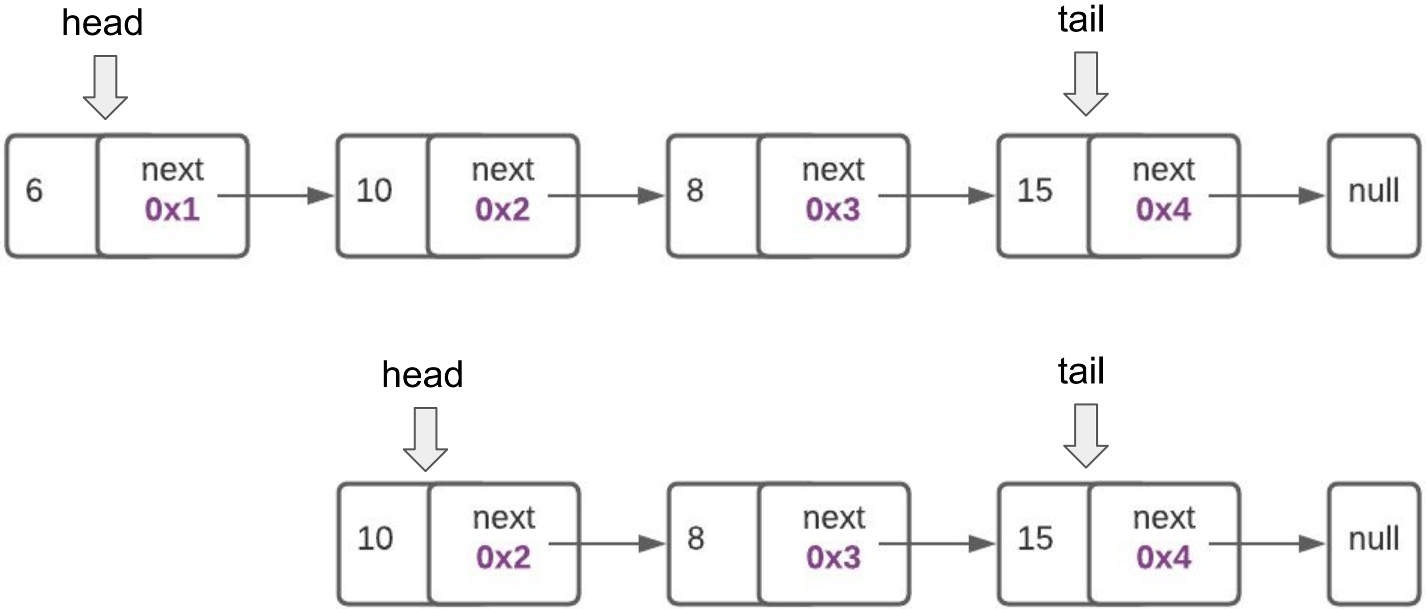
Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



Nodes deletion

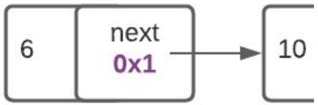
- We typically might need 3 standard deletion for nodes
    - Delete the first node
    - Delete the last node
    - Delete the nth node or node with a value
- You know enough to code them by yourself
    - Think in the different cases for each
    - Draw the list before
    - Draw the list per after each step

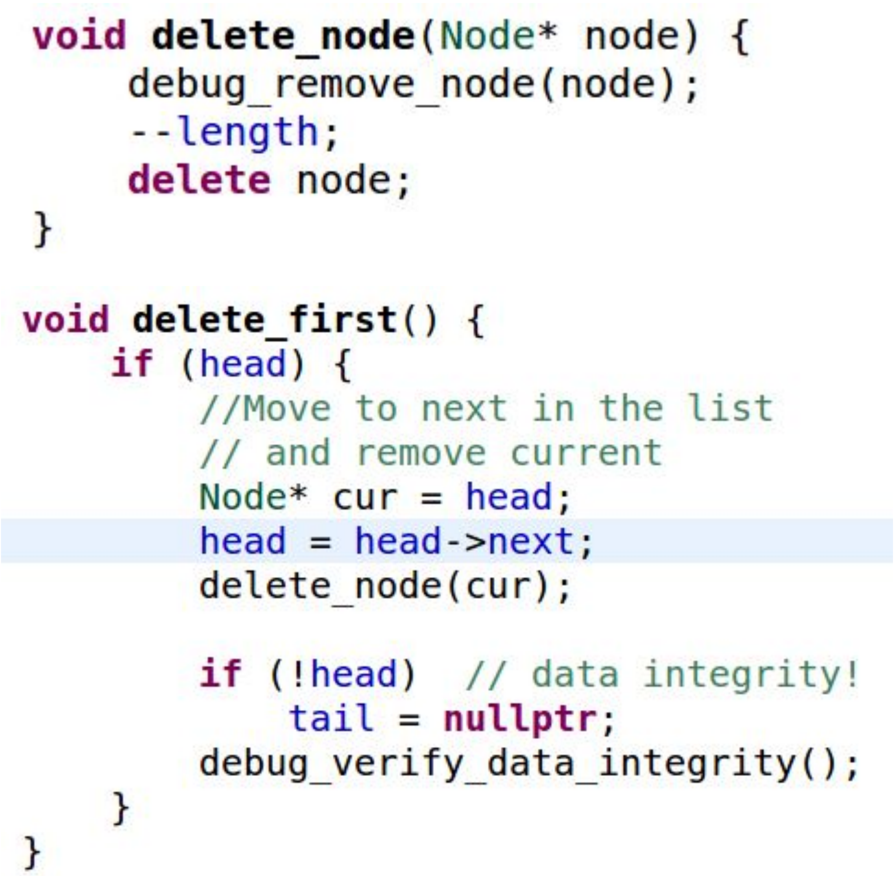
Delete first node



Delete first node

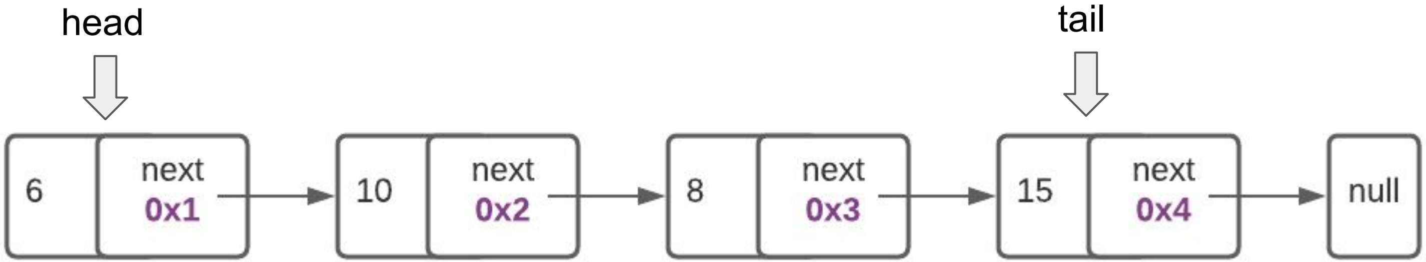
- Simply we need to make the head->next as the new head
- But take pointer first on head to delete it
- Concerns
  - No deletion for empty list
  - If list is empty, make sure tail is null

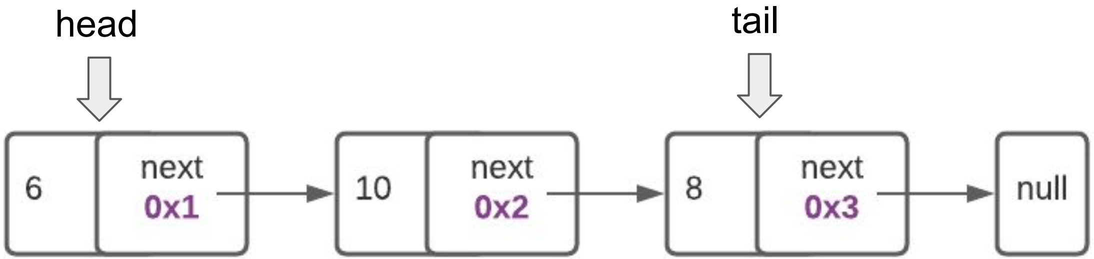




```
void delete_node(Node* node) {
    debug_remove_node(node);
    --length;
    delete node;
}

void delete_first() {
    if (head) {
        //Move to next in the list
        // and remove current
        Node* cur = head;
        head = head->next;
        delete_node(cur);

        if (!head)  // data integrity!
            tail = nullptr;
        debug_verify_data_integrity();
    }
}
```

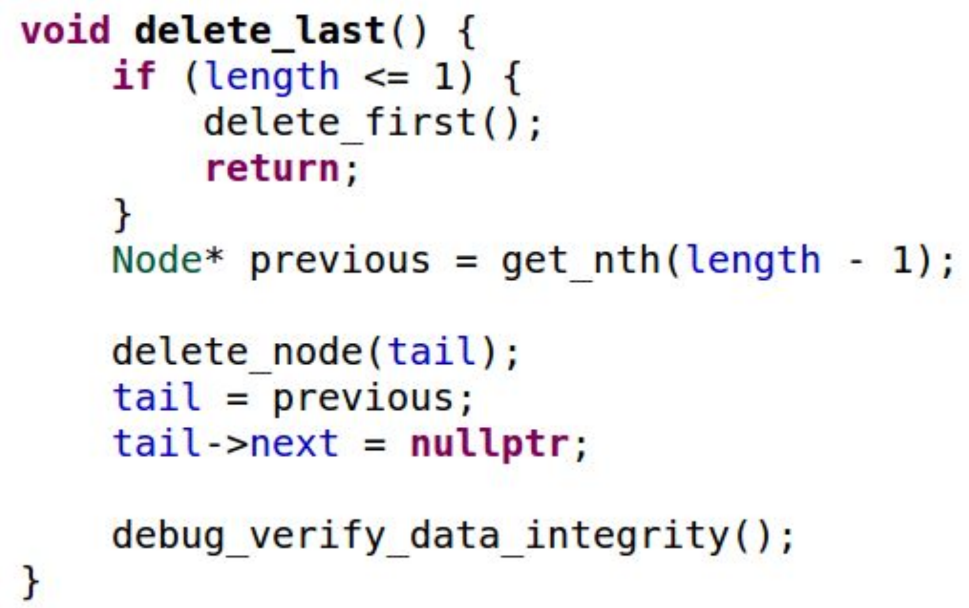
Delete last node





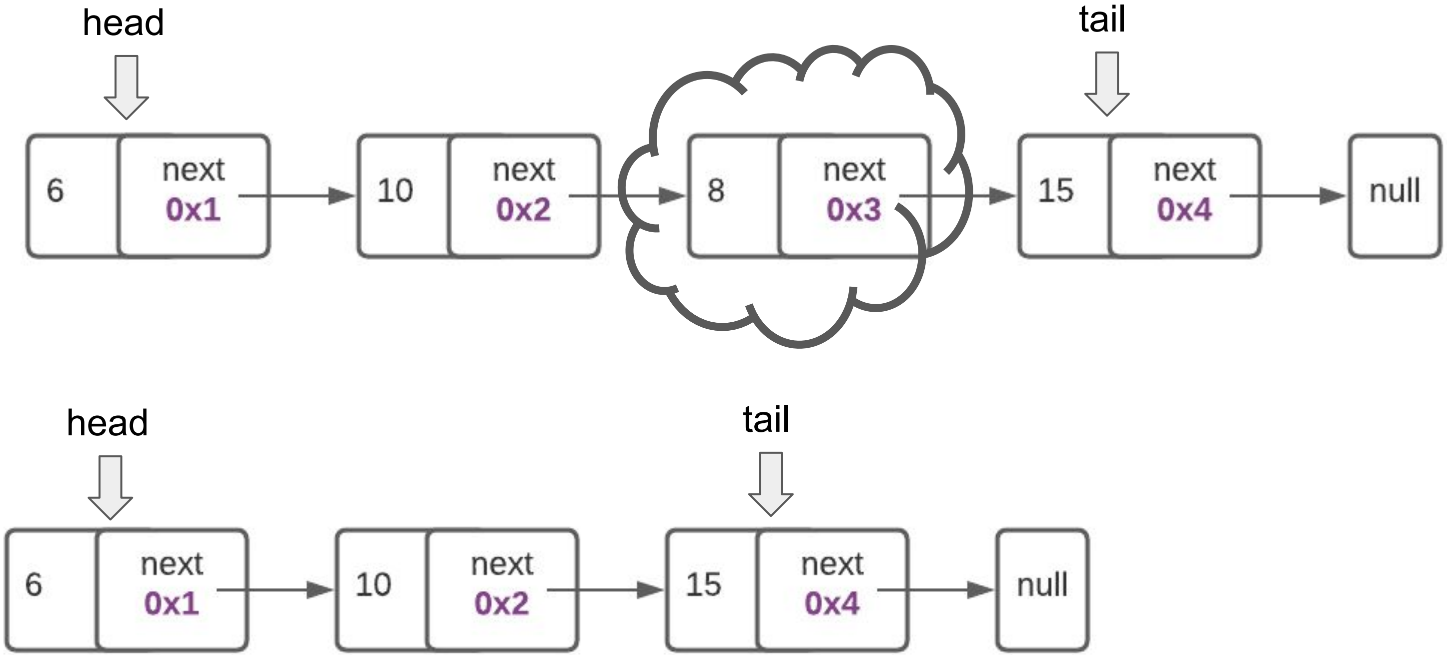
Delete last node

- We need make the node before the tail as the new tail
- How to get it?
    - Simple loop can find it
    - A better trick: use get_nth
    - Length-1 is the node before tail
- Also observe delete_last
    - We utilized 2 old functions!
- Delete old tail and make new one



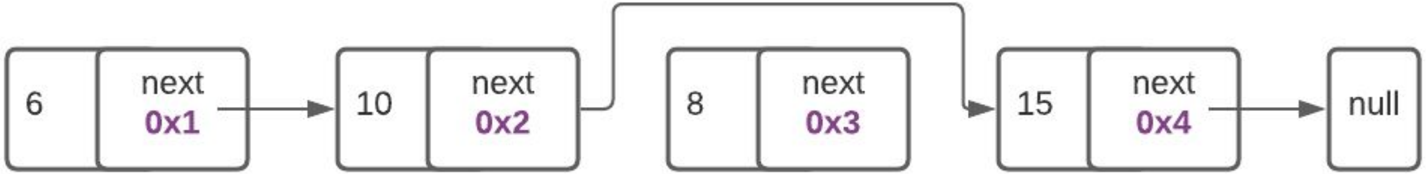
```
void delete_last() {
    if (length <= 1) {
        delete_first();
        return;
    }
    Node* previous = get_nth(length - 1);

    delete_node(tail);
    tail = previous;
    tail->next = nullptr;

    debug_verify_data_integrity();
}
```

Delete nth node (e.g. 3)

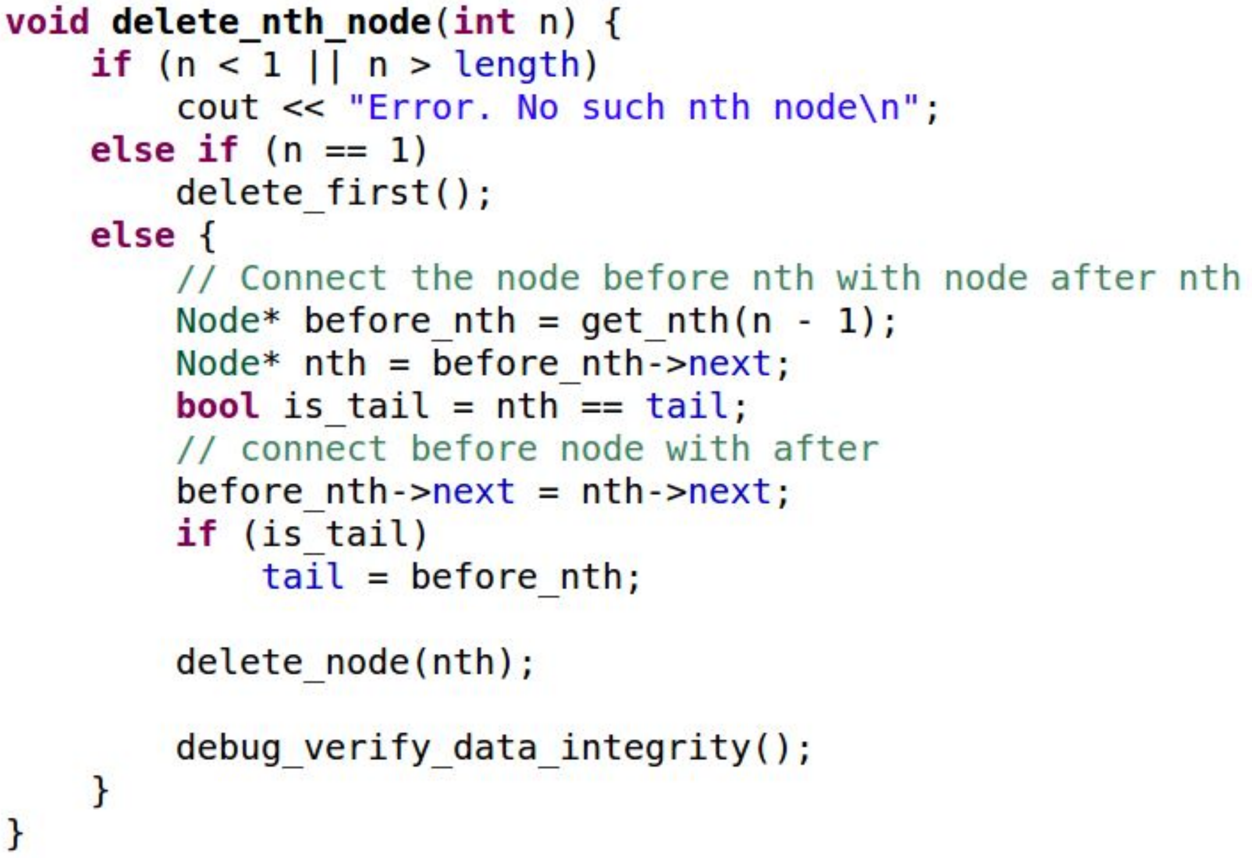


Delete nth node

- First node is a special case
- All what we need is to link (n-1) node with (n+1) node
  - We need the node before nth and node after it
  - We need to connect them together
- Code trick
  - If we know the nth node, we easily now the n+1. But not the n-1 node!
  - Better: get the n-1 node. Then next is (n) and next->next is (n+1)



Delete nth node



```
void delete_nth_node(int n) {
    if (n < 1 || n > length)
        cout << "Error. No such nth node\n";
    else if (n == 1)
        delete_first();
    else {
        // Connect the node before nth with node after nth
        Node* before_nth = get_nth(n - 1);
        Node* nth = before_nth->next;
        bool is_tail = nth == tail;
        // connect before node with after
        before_nth->next = nth->next;
        if (is_tail)
            tail = before_nth;

        delete_node(nth);

        debug_verify_data_integrity();
    }
}
```

Tip

- In most of medium/hard challenges in linked lists, we need to relink nodes
    - Delete nth node is an example for that
- You need to determine which links will be changed
    - From - To
- Order matters, especially if you are deleting
    - E.g. take its next first before deleting it
- Draw. Draw. Draw. Draw. Draw EVERY step

Linked List ADT

- The ADT of (Linked) List is a collection of data nodes accessed **sequentially**.
    - Its main functionalities (interface): Add/Delete, Start, Next, Length functionalities
- We learned how to implement the list in C++ using a linked list data structure of (*head, tail and length*) and several variants to insert/delete items
- One clear disadvantage comparing with arrays is **no random access** O(1), but we gain flexibility in memory growth (no need for vector reallocations)

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."