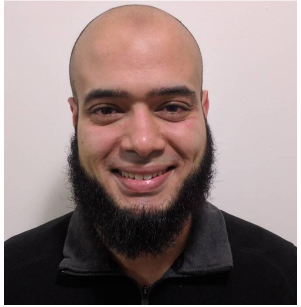
# Data Structures

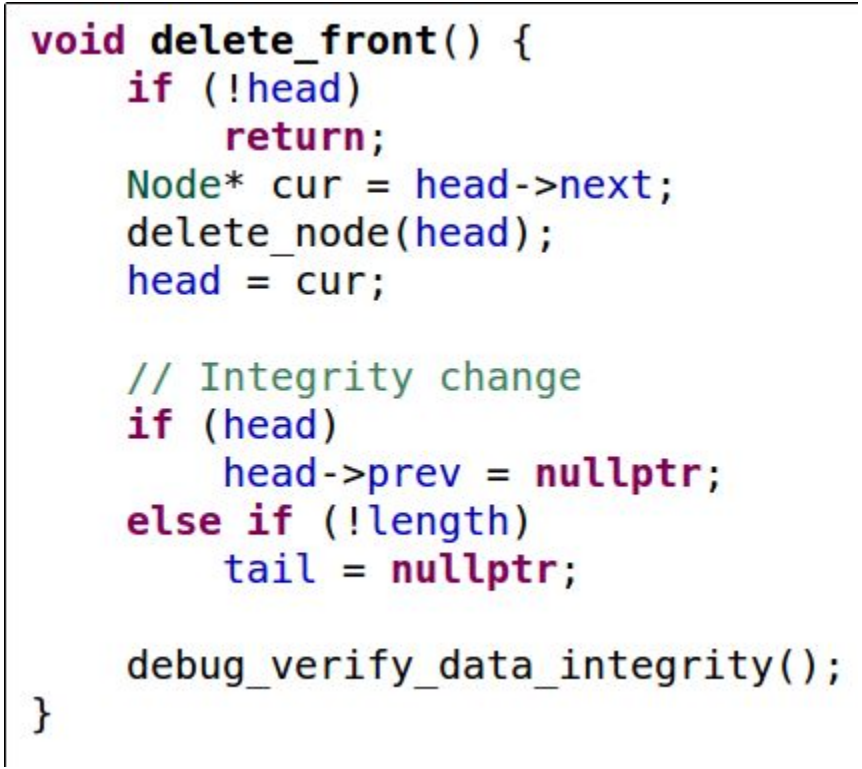
# DLL Deletion

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

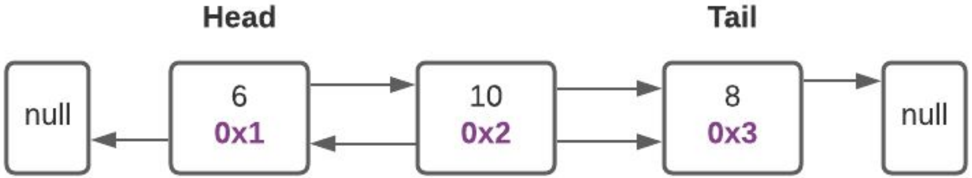
*Artificial Intelligence & Computer Vision Researcher*
*PhD* from Simon Fraser University - Canada
*Bachelor / Msc* from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



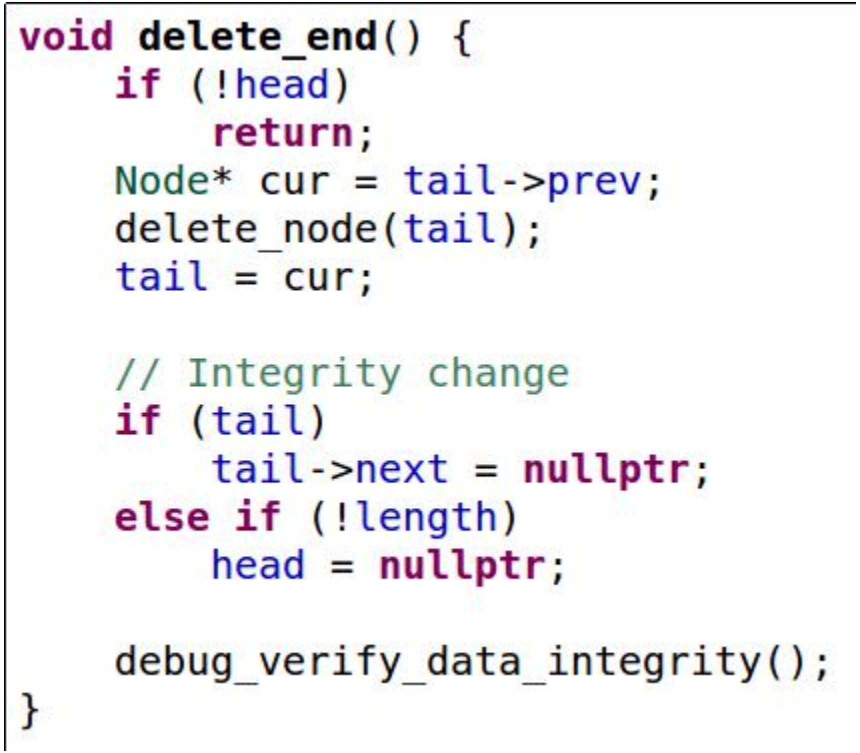
# Delete Front



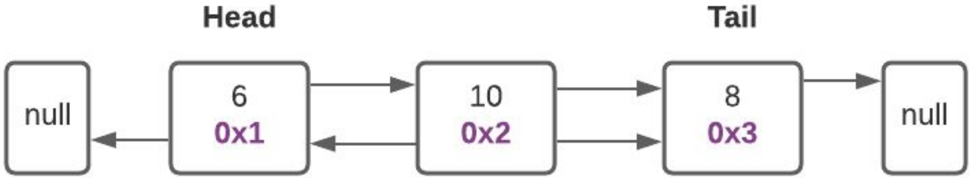
```
void delete_front() {
    if (!head)
        return;
    Node* cur = head->next;
    delete_node(head);
    head = cur;

    // Integrity change
    if (head)
        head->prev = nullptr;
    else if (!length)
        tail = nullptr;

    debug_verify_data_integrity();
}
```



# Delete End

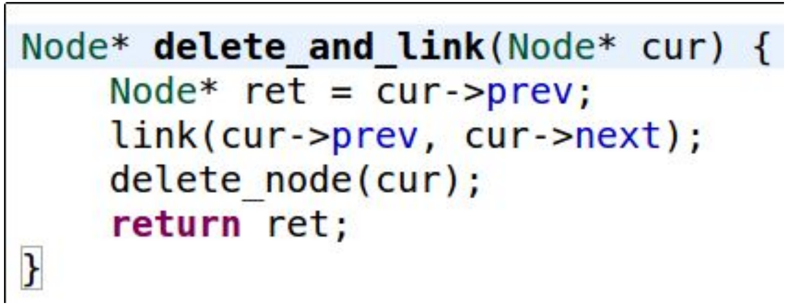


```
void delete_end() {
    if (!head)
        return;
    Node* cur = tail->prev;
    delete_node(tail);
    tail = cur;

    // Integrity change
    if (tail)
        tail->next = nullptr;
    else if (!length)
        head = nullptr;

    debug_verify_data_integrity();
}
```

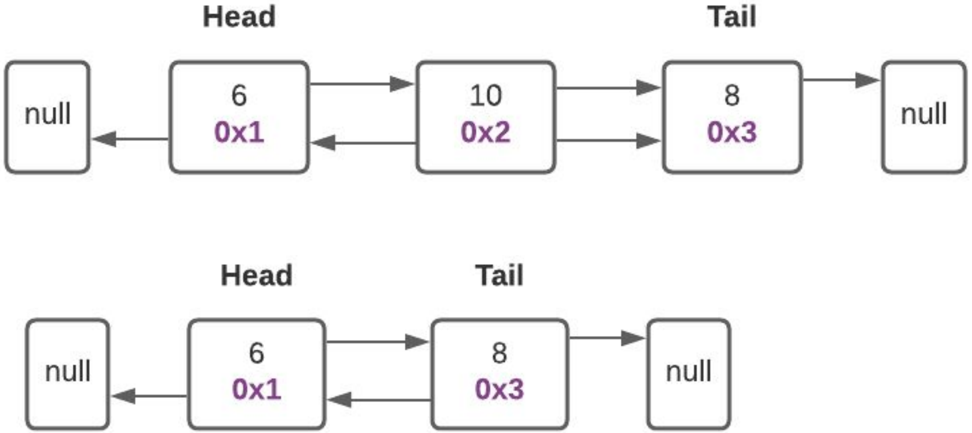


# delete_and_link utility

- Given a node, connect its previous and next, and then delete it
  - Return the previous node
- Let’s delete node at 0x2 (value 10)

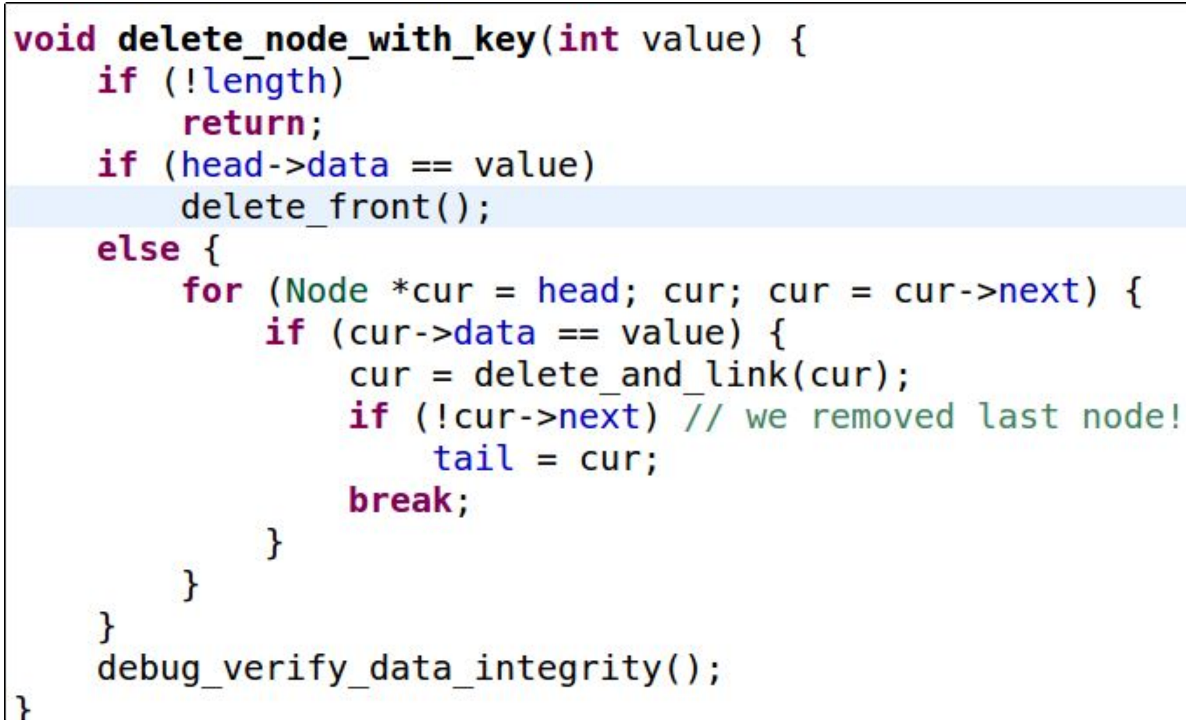


```
Node* delete_and_link(Node* cur) {
    Node* ret = cur->prev;
    link(cur->prev, cur->next);
    delete_node(cur);
    return ret;
}
```



# Delete node with key

- Similar logic to SLL
- Delete returns for us the previous of it
- Special handling if last element is removed



```
void delete_node_with_key(int value) {
    if (!length)
        return;
    if (head->data == value)
        delete_front();
    else {
        for (Node *cur = head; cur; cur = cur->next) {
            if (cur->data == value) {
                cur = delete_and_link(cur);
                if (!cur->next) // we removed last node!
                    tail = cur;
                break;
            }
        }
    }
    debug_verify_data_integrity();
}
```

“Acquire knowledge and impart it to the people.”

“Seek knowledge from the Cradle to the Grave.”