Data Structures

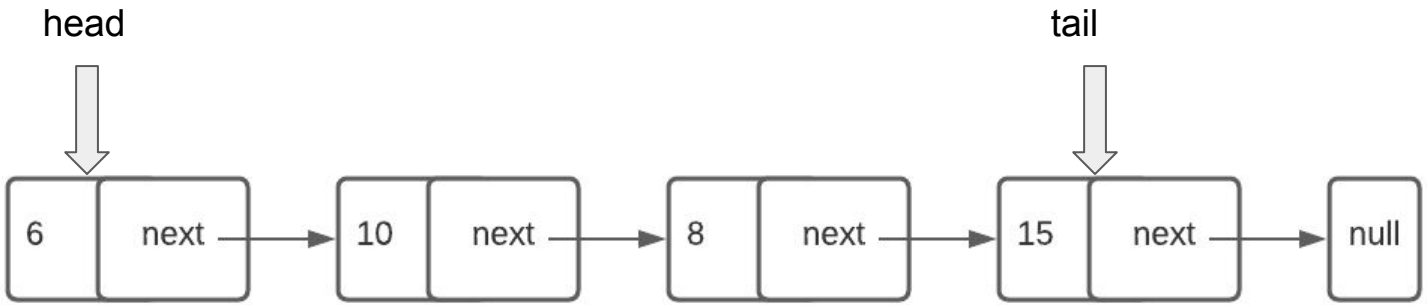
Singly Linked List

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)

Singly Linked List

- Linked List is a **sequence of nodes**, where each node contains data and link to the next node creating a dynamic connected chain of values
  - We can easily expand its data with a new element in **O(1) memory**
  - Several further operations are possible and also limitations
- The first node is called **head** and the last one (**optionally**) is called **tail**



Singly Linked List: Data Structure

- The linked list data structure is simply 2 nodes head and tail
- Several operations can be supported to manipulate content

```
struct Node {
    int data {};
    Node* next {};
    Node(int data) : data(data) {}
};

class LinkedList {
private:
    Node *head { };
    Node *tail { };
public:
    void print() {
    void insert_end(int value) {
};
```

Singly Linked List: print

- We already implement this before. 2 changes:
- We will move our code, but:
- We should keep the head all time looking to begin
- To handle that: take copy of it
- Tip: Don’t corrupt your DS

```
void print() {
    // DON'T change head itself.
    //      You will lose it
    Node* temp_head = head;
    while(temp_head != nullptr) {
        cout<<temp_head->data<<" ";
        temp_head = temp_head->next;
    }
    cout<<"\n";
}
```

Singly Linked List: insert_end

- In the previous lectures, we created and inserted the nodes manually
- Insert_end member function should do this automatically
- Take 10 minutes try to implement it by yourself.
- Clearly 2 cases:
- Case 1: the head/tail is null initially
- Case 2: we have a list of items and want to add
    - Tip. DRAW the list BEFORE and AFTER

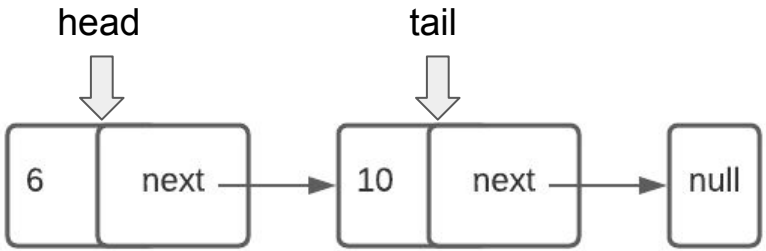
```
LinkedList list;
list.insert_end(6);
list.insert_end(10);
list.insert_end(8);
list.insert_end(15);
list.print();
// 6 10 8 15
```

Singly Linked List: insert_end

- Initially, we create a node with the requested value
- Case 1: the head/tail is null initially
  - Just set head = tail = this new node
- Case 2: 1+ item in the list
  - Head won’t be changed
  - We need to link tail to the new item
  - Then set tail to the new item, as it is the tail now

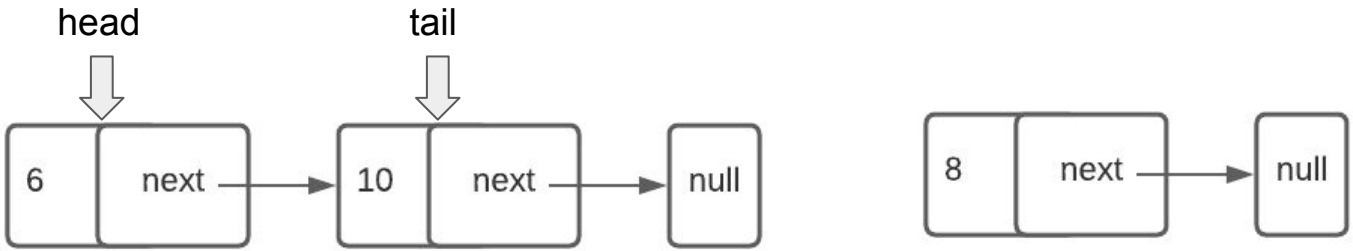
Singly Linked List: insert_end

- Assume current list is (6, 10)
- Our goal is to add value 8



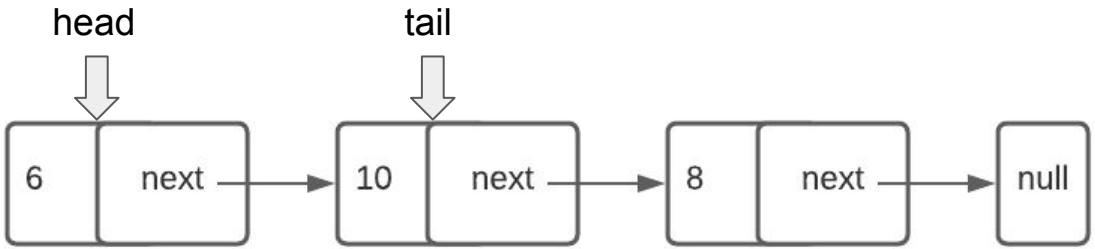
Singly Linked List: insert_end

- Assume current list is (6, 10)
- Our goal is to add value 8
- First: let's create a node with value 8 (item)



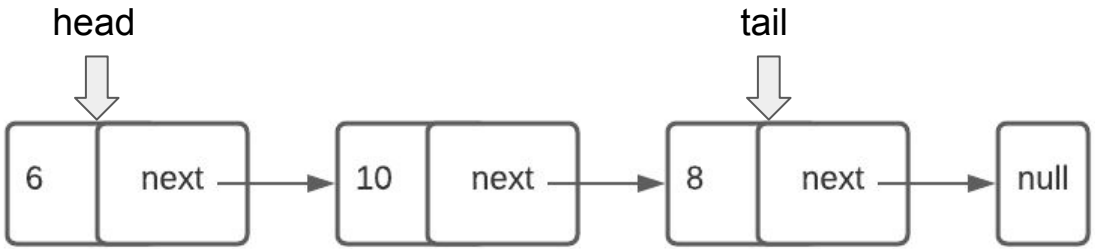
Singly Linked List: insert_end

- Assume current list is (6, 10)
- Our goal is to add value 8
- Second, let’s link tail (value 10) with the new node (8)
  - tail->next = item;



Singly Linked List: insert_end

- Assume current list is (6, 10)
- Our goal is to add value 8
- Third, Update the tail to look to the new last node
  - tail = item;



Singly Linked List: insert_end

- Now, we can code this logic in this simple way!
- General note: if we did not support **destructor** to remove created memory, we will have a *memory leak*

```
void insert_end(int value) {
    Node* item = new Node(value);

    if (!head)
        head = tail = item;
    else {
        tail->next = item;
        tail = item;
    }
}
```

Easier coding

- Many courses use only the **head** pointer without tail
  - With tail, some problems are shorter to code
  - Sometimes, it is a good challenge to try solving a problem without a tail
- Int length
  - Introduce also length variable
  - With each insertion increment it and with deletion decrease it
  - It will make many coding checks much easier
  - Common mistake
    - As we code several insert/delete functions,
      You forget to maintain length variable

```
class LinkedList {
private:
    Node *head { };
    Node *tail { };
    int length = 0;
```

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."