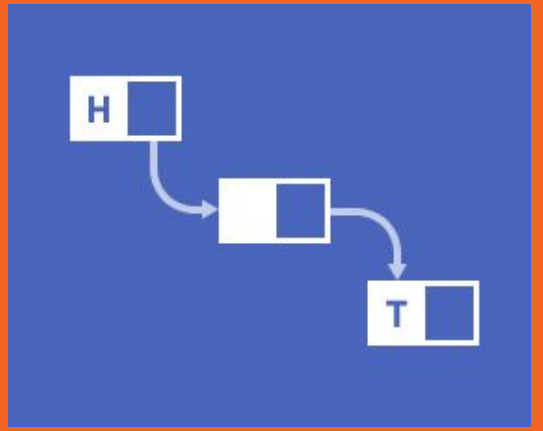
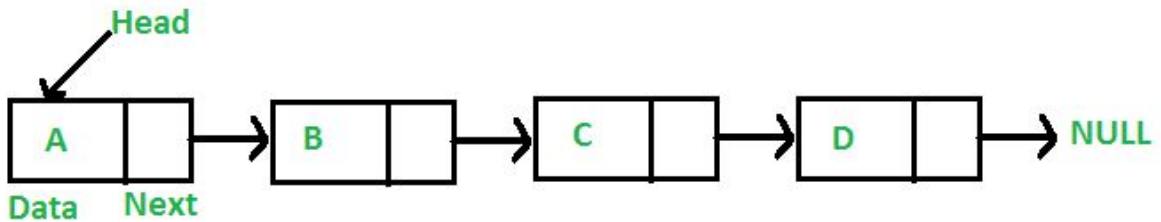
Linked Lists

Maggie Shi and Alex Xu



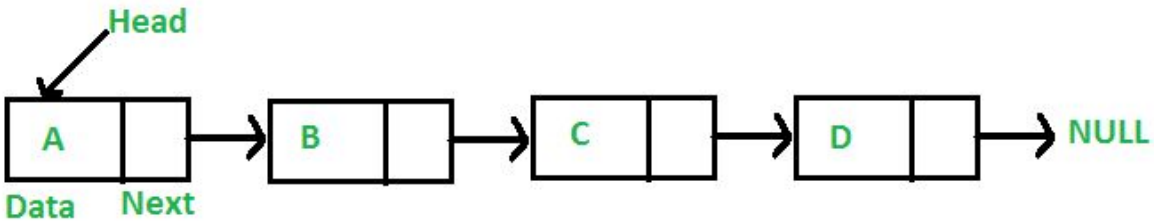
What is a linked list?

- An alternative to an array
- Also stores a collection of values
  - Could be generalized to int, string, etc.
- But very different in how it's structured/used!



What is a linked list?

- Linked lists have two components: nodes and pointers
- The pointer to the first node in the list is called the “head”
  - Typically what we’re “given”
- Each node contains a pointer that points to the next node in the list
- The last node in the list points to nullptr



Check for Understanding

- How do we know when we're at the end of the linked list?
- What would a Node struct look like?

Check for Understanding

- How do we know when we're at the end of the linked list?
  - When the current node points to nullptr!
- What would a Node struct look like?

```
class Node {

public:

    int value; // could be any type you want

    Node* next;

};
```

Keep in mind

- Some things you CAN’T do:
  - Random access, e.g. arr[i]
  - Find the current length trivially, e.g. arr.length

Iterating through a linked list

Pseudocode:

```
iterate(head):
    p = head // temp pointer
    while p != null: // last node points to null
        // p is now the next node in the list
        p = p.next
```

Exercise #1

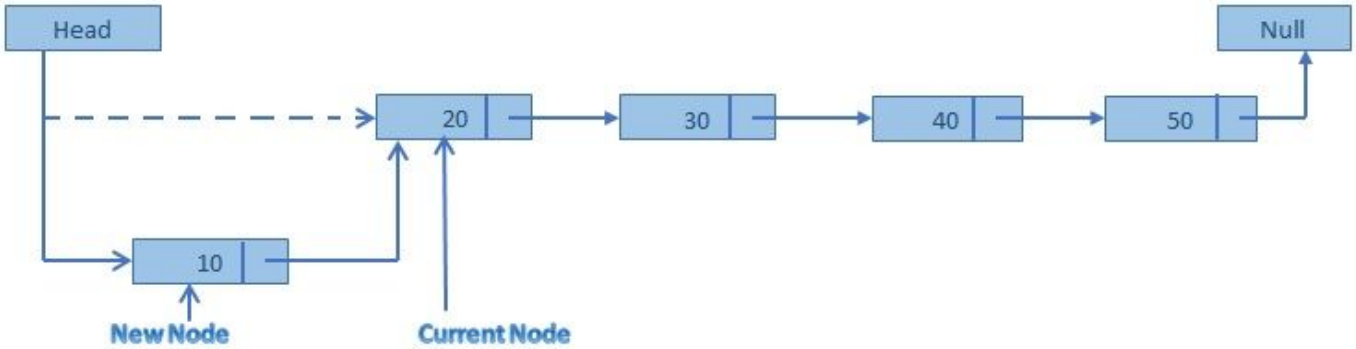
- Come up with a function

  ```
  bool contains(Node* head, int val)
  ```

  which checks if a linked list contains a certain value

```
bool contains(Node* head, int val) {
    Node* curr = head;
    while (curr != nullptr) {
        if (curr->value == val)
            return true;
        curr = curr->next;
    }
    return false;
}
```

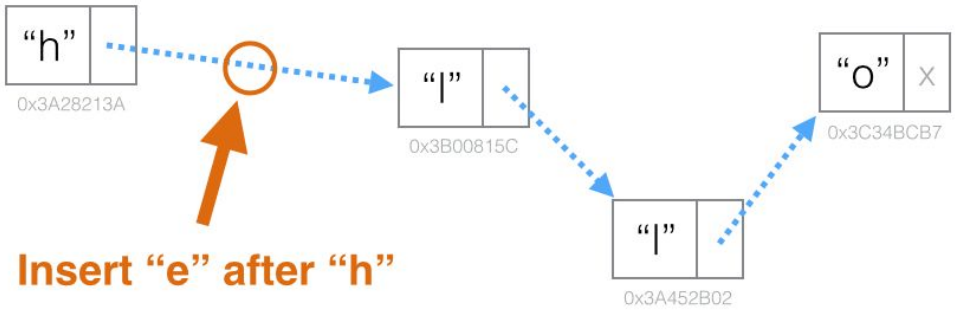
Inserting a node to the front



Inserting a node to the front

```
void insertToFront(int val) {

    Node* p;

    p = new Node;

    p->value = val;

    p->next = head;

    head = p;

}
```

Inserting a node in the middle

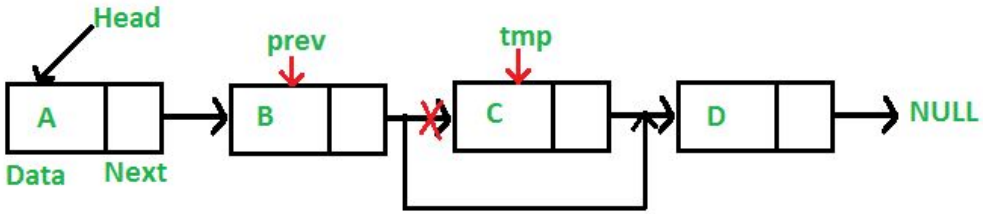


Inserting a node in the middle

```
// want to insert new node before target value
void insertInMiddle(int val, int target) {
    Node* curr = head;

    while (curr->next != nullptr) {
        if (curr->next->value == target)
            break;
    }

    // now curr is the node before the target node
    Node* p = new Node(val);
    p->next = curr->next;
    curr->next = p;
}
```

Deleting a node

- Use two pointers (“fast” and “slow”), one a step ahead
- Iterate through the list with both at the same time
- When “fast” reaches the target node
- Change “slow”’s next pointer to point to “fast”’s next pointer



Now...

- What’s an advantage of using a linked list over an array?
- What’s an advantage of using an array over a linked list?

Exercise #2

Write a function `combine` that takes in two sorted linked lists and returns a pointer to the start of the resulting combined sorted linked list. You may write a helper function to call in your function `combine`.

Other types

- Doubly-linked list - each node now also has a “previous” pointer
  - How would the Node struct look now?
- Circular list - the last node’s next pointer points to the head