SCC.121: Fundamentals of Computer Science
Linked Lists

Amit Chopra

amit.chopra@lancaster.ac.uk

What is a List?

- A list is an Abstract Data Type that stores a *set* of items in a linear order.
  - No duplicates
  - (Assume no duplicates are given by the user)
    - (So the list implementation doesn’t have to do duplicate checking)
- Widely useful
    - Shopping list
    - List of friends on social media
    - List of student records
    - To do list

Doubly Linked List

```
Element {
    Item data
    Element next, prev //pointers to Element
}
```

- `Add(L, Element e)`
  - Add Element e to the list L
- `remove(L, e)`
  - Remove e from L
- `Element search(L, Item k)`
  - Returns pointer to Element containing k if k is present in L, else nil
- `int size(L)`
  - `Returns the count of items in L`

Doubly Linked List

```
//Initially head of any List is nil, indicating that list is empty
List() {
    Element head = nil
}
```

Doubly Linked List

Graphically, an element is

prev	data	next

`/` means the value of the pointer is nil

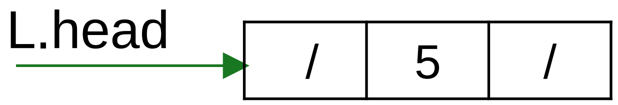
/	data	next

means prev is nil

- add(L,

	5	

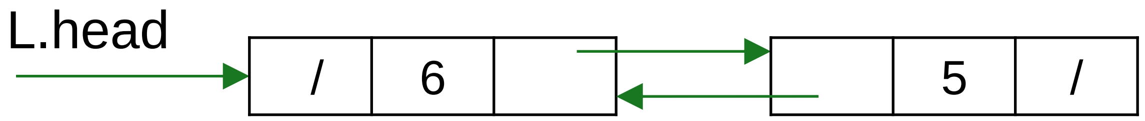
)



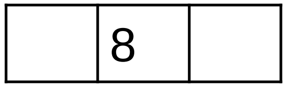
- add(L,

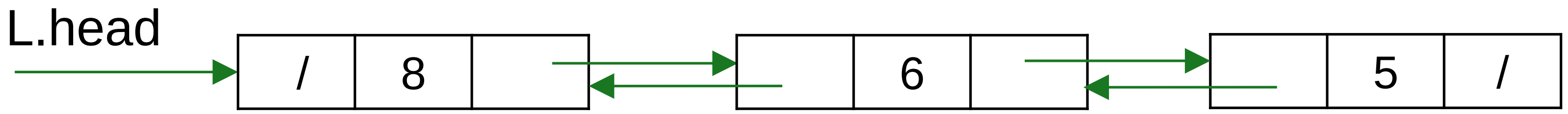
	6	

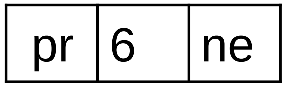
)

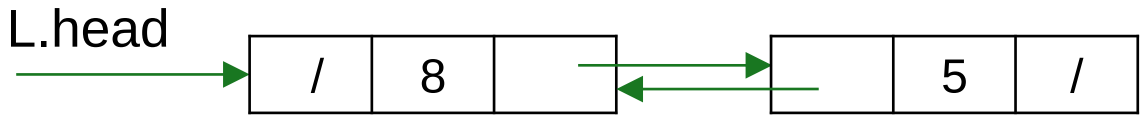


(continued)

- add(L, )



- remove(L, )



Implementation

```
search(L, k){
      p = L.head
      while(p != nil && p.data != k)
            p = p.next
      return p
}

add(L,e) {
      e.next = L.head //Adding at the front
      if(L.head != nil)
            L.head.prev = e
      L.head = e
      e.prev = nil
}
```

Continued

```
remove(L, e){
      if (e.prev != nil)  //Not first element
            e.prev.next = e.next
      else
            L.head = e.next
      if (e.next != nil)  // Not last element
            e.next.prev = e.prev
}
```

Singly Linked List

- No prev pointer
- Unlike doubly linked list,
  - Can only be traversed in forward direction
    - An inconvenience, e.g., when printing a list in reverse order
  - Removing always requires traversal

Many Variations

- Insert and remove at particular position in list
- Remove by data
- Iterators: getFirst() and getNext()
- E.g., *tail* in doubly linked list may be used to point to end of list and make reverse traversal even more convenient
- Sentinels (special markers) may be used to simplify code