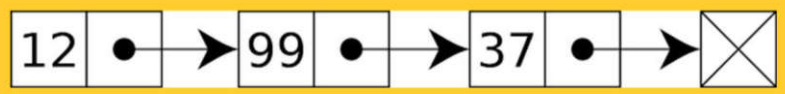
ITD62-124 Data Structure

List



Outline:

- Introduction
- Singly linked list
- Doubly linked list
- Circular list

Introduction

- Arrays and Linked lists are similar because they both store collections of data
- Arrays allocate the memory for all its elements in one block of memory
- Disadvantages of Arrays:
  - ✓the size of the array is fixed
  - ✓difficult to insert elements inside the array

Introduction

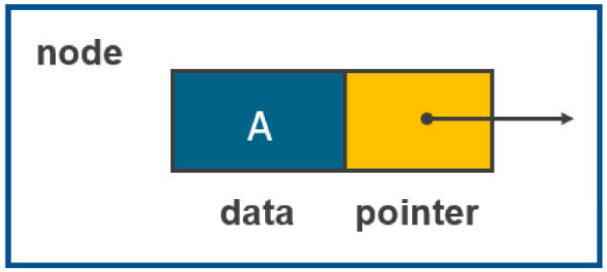
- A Linked list is a collection of nodes storing data and links to other nodes
- Nodes can be located anywhere in memory
- Linked lists are dynamic, the length of a list can increase or decrease as necessary

Introduction

- Linked lists can be maintained in sorted order by inserting or deleting an element at the proper point in the list
- Advantages of Linked lists:
  - ✓a linked list can easily increase and decrease the size
  - ✓easy and fast insertions and deletions

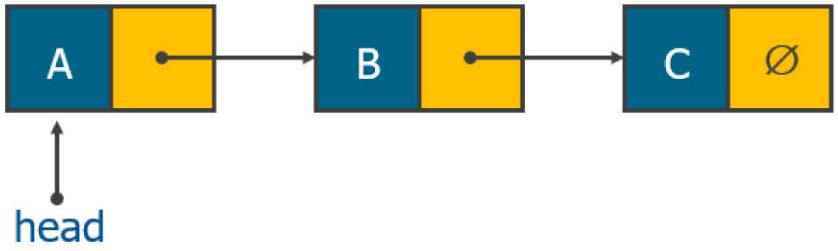
Introduction

- A linked list is a series of connected nodes
- Each node contains at least
  - ✓ A piece of data (any type)
  - ✓ Pointer to the next node in the list



Introduction

- head: pointer to the first node
- The last node points to `NULL`



Introduction

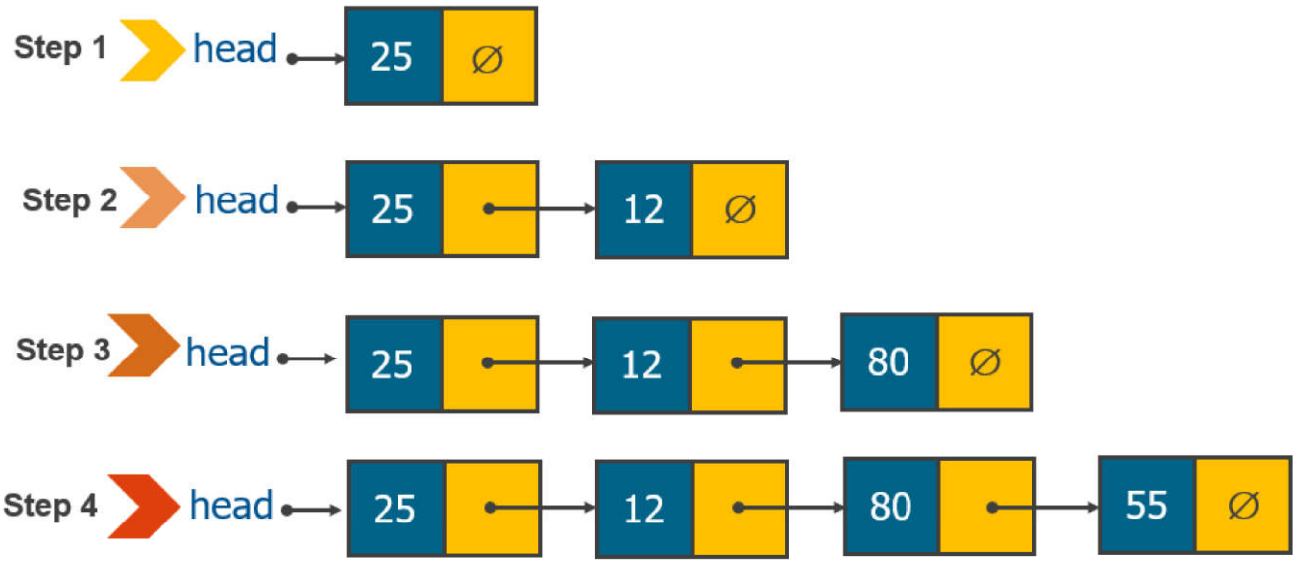
- Type of Lists:
  - ✓Singly linked list
  - ✓Doubly linked list
  - ✓Circular list

Singly Linked Lists

- A node has a link only to its successor
- Each node includes two data members:
  - ✓ The `info` member is used to store information
  - ✓ The `next` member is used to link nodes to form a linked list

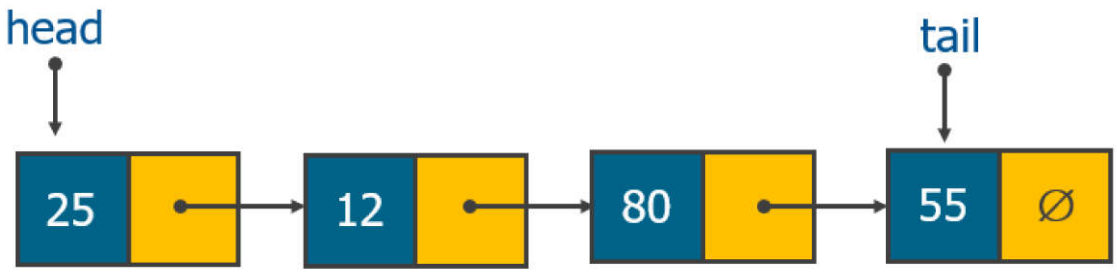
Singly Linked Lists

- Example: create a singly linked list (25, 12, 80, 55)



Singly Linked Lists

- `head` is a pointer to the first node of a list
- `tail` is a pointer to the last node of a list
- Example:



Singly Linked Lists Operations

- Insertion:

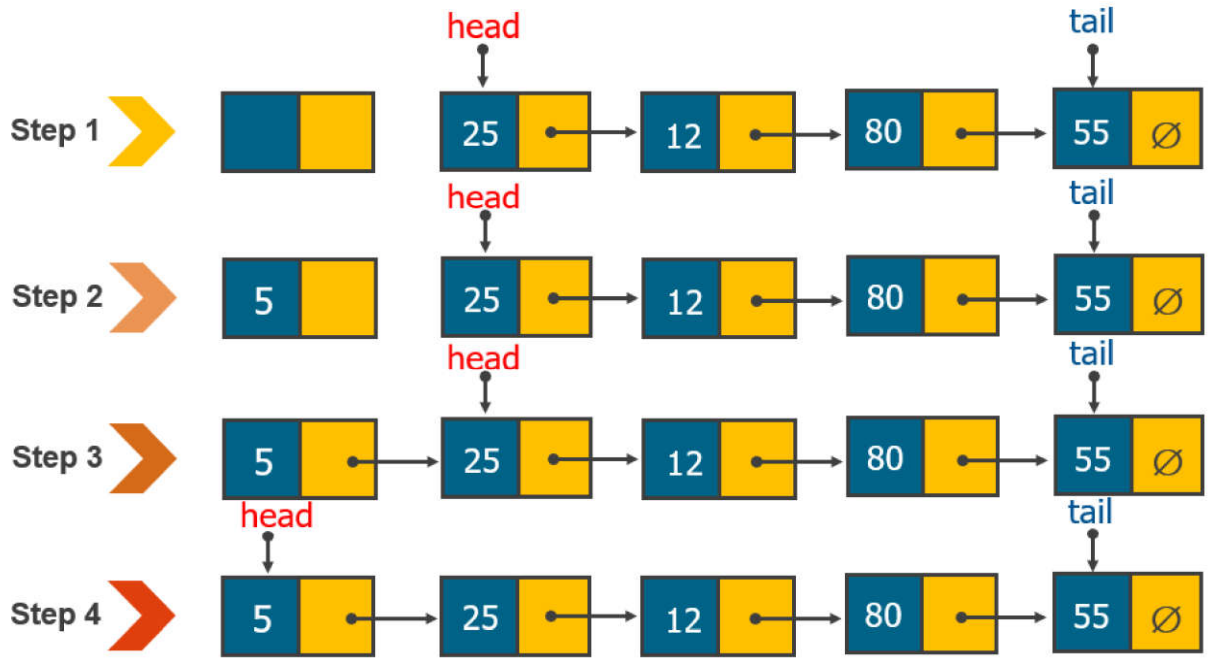
✓ inserting a node at the beginning of a singly linked list
✓ inserting a node at the end of a singly linked list

Singly Linked Lists Operations

- Inserting a node at the beginning of a singly linked list
  - ✓Create an empty node
  - ✓Initialize the node's `info` member
  - ✓The `next` member becomes a pointer to the first node on the list
  - ✓`head` is updated to become the pointer to the new node

Singly Linked Lists Operations

- Example: insert node 5 at the beginning of a singly linked list

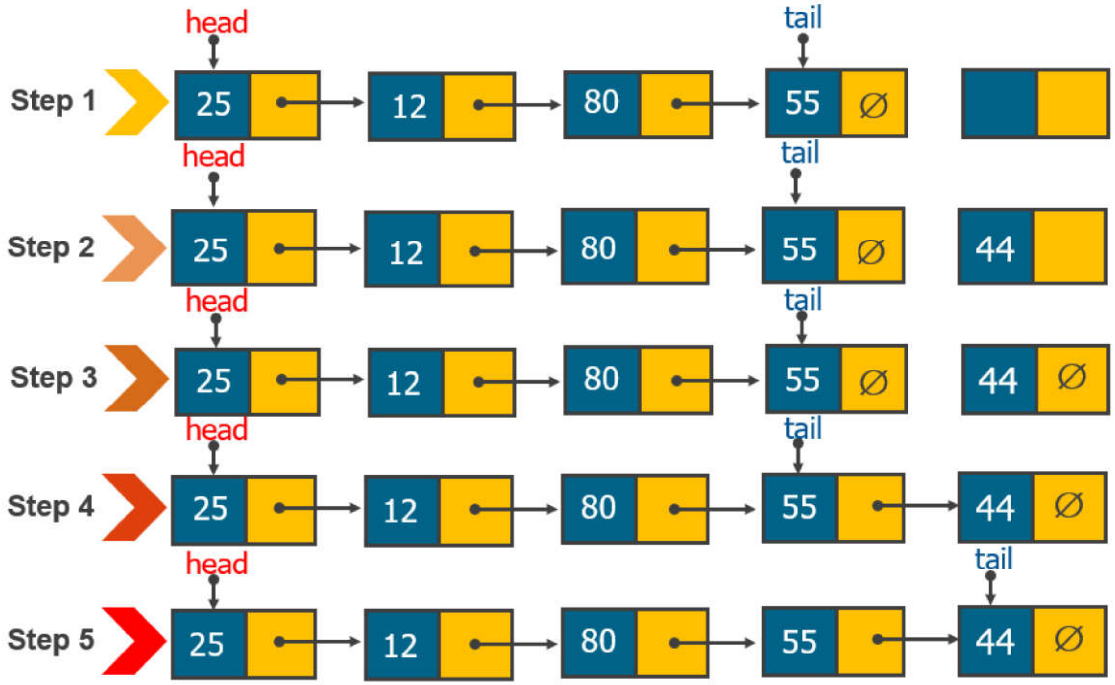


Singly Linked Lists Operations

- Inserting a node at the end of a singly linked list
  - ✓Create an empty node
  - ✓Initialize the node's `info` member
  - ✓The `next` member is set to `null`
  - ✓The `next` member of the last node of the list becomes a pointer to the new node
  - ✓`tail` is updated to become the pointer to the new node

Singly Linked Lists Operations

- Example: insert node 44 at the end of a singly linked list



Singly Linked Lists Operations

- Deletion:
  - ✓deleting a node at the beginning of a singly linked list
  - ✓deleting a node from the end of a singly linked list
  - ✓deleting a node from a singly linked list

Singly Linked Lists Operations

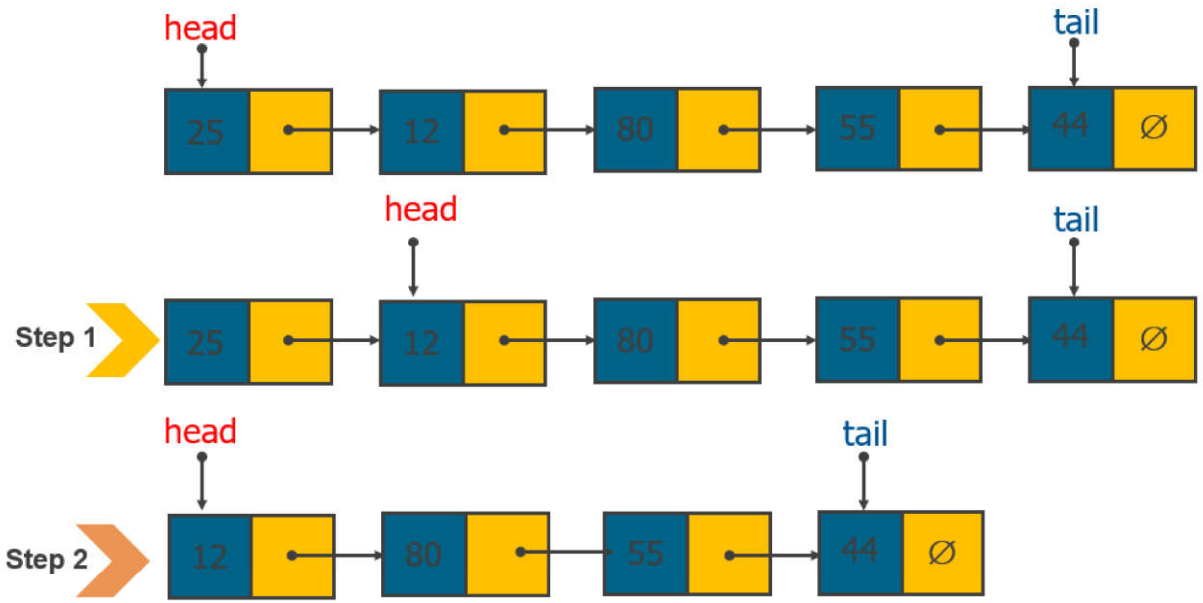
- Deleting a node at the beginning of a singly linked list

✓`head` is updated to become the pointer to the second node

✓The second node becomes the first node of the list

Singly Linked Lists Operations

- Example: delete node 25 from a singly linked list



Singly Linked Lists Operations

- Deleting a node from the end of a singly linked list

✓`tmp` first refers to the head node

✓update the value of `tmp` where `tmp = tmp->next` until `tmp->next` is the last node of the list

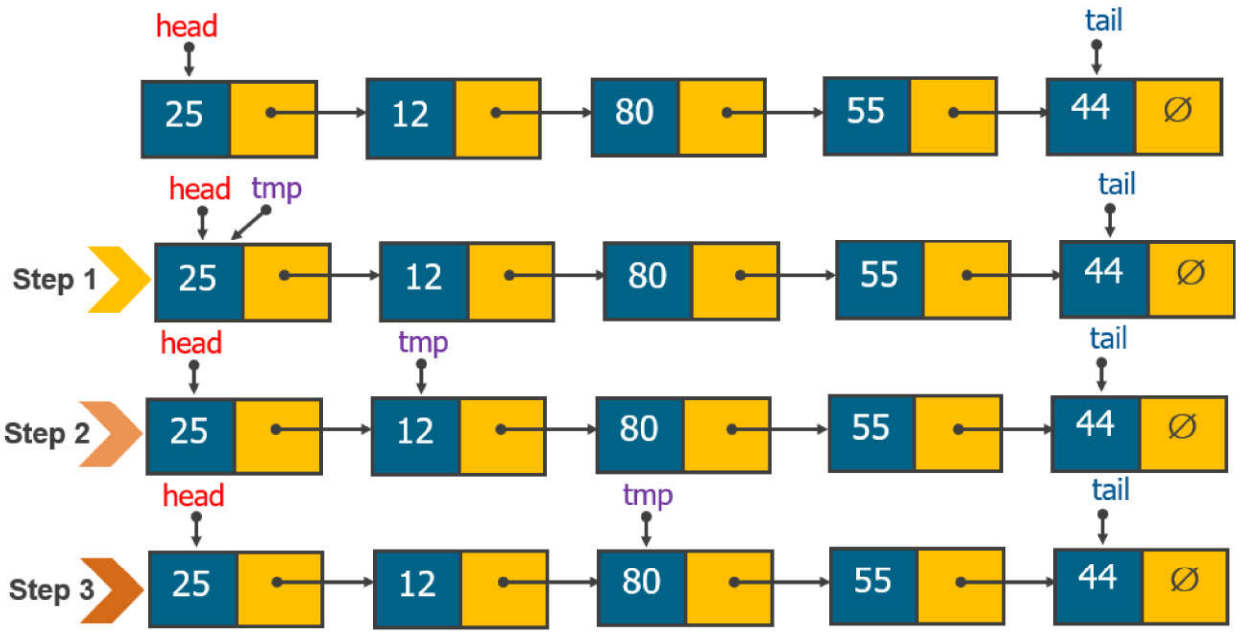
✓delete last node

✓`tail` is updated to become the pointer to the node currently pointed to by `tmp`

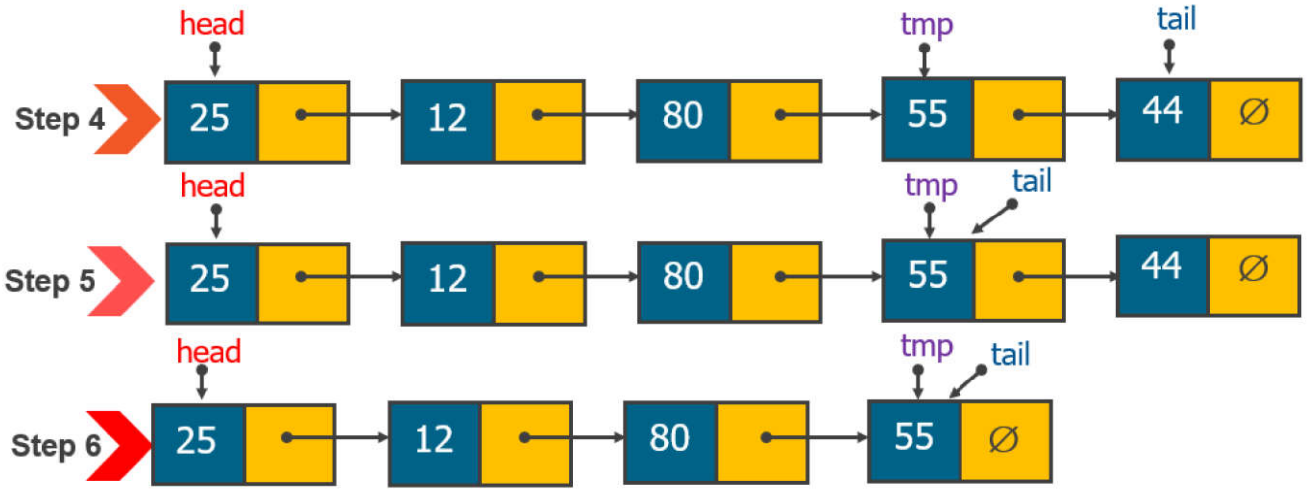
✓The `next` member of the last node of the list is set to `null`

Singly Linked Lists Operations

- Example: delete node 44 from a singly linked list



Singly Linked Lists Operations



Singly Linked Lists Operations

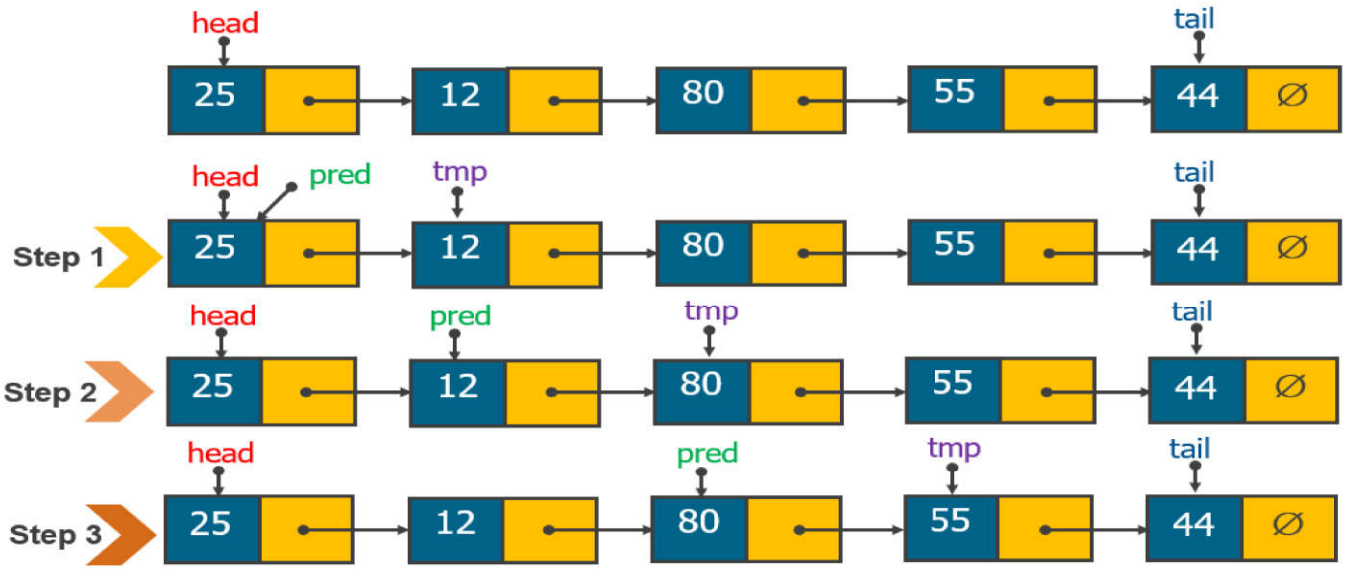
- Deleting a node from a singly linked list

✓a node has to be located

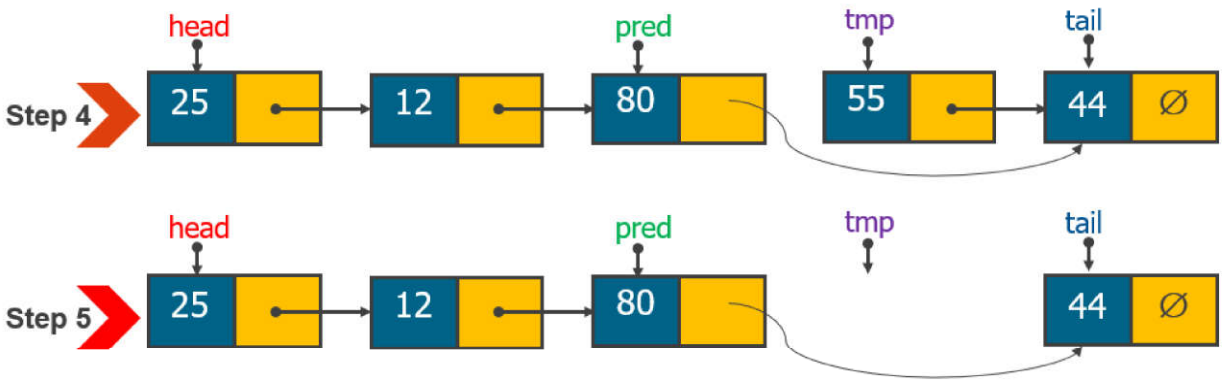
✓linking the predecessor of this node directly to its successor

Singly Linked Lists Operations

- Example: delete node 55 from a singly linked list



Singly Linked Lists Operations



Singly Linked Lists Operations

- Search:

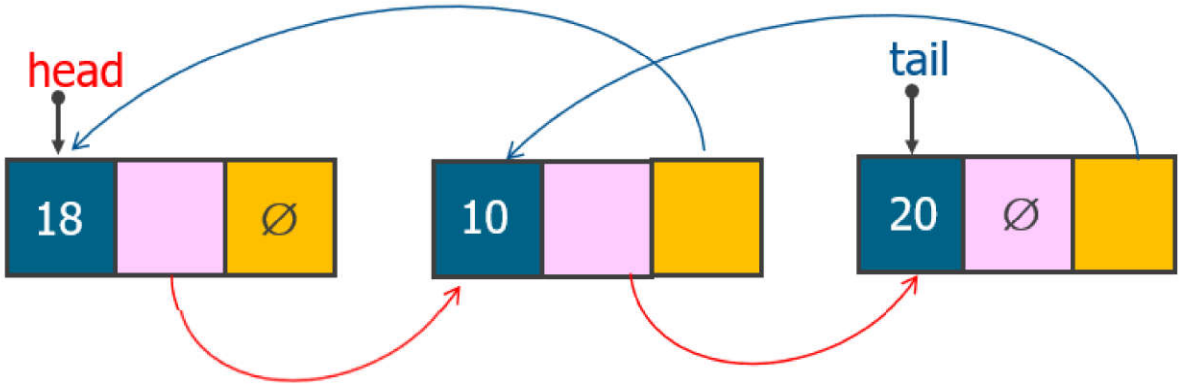
✓scan an existing list to learn whether a number is in it

✓use `tmp` to go through the list starting from the head node

✓compare the number stored in each node to the number being sought

Doubly Linked Lists

- A node has two pointers, a successor and a predecessor
- Example:



Doubly Linked Lists Operations

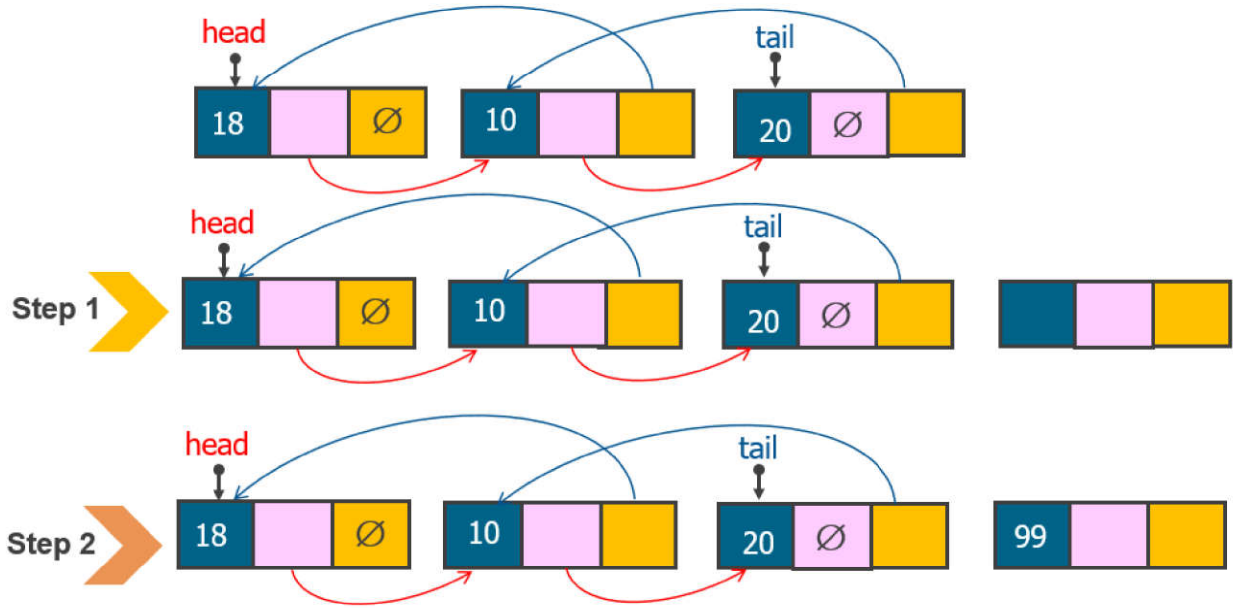
- Insert a node at the end of the doubly linked list
- Insert a node into a doubly linked list
- Delete a node from the end of the doubly linked list
- Delete a node from a doubly linked list

Doubly Linked Lists Operations

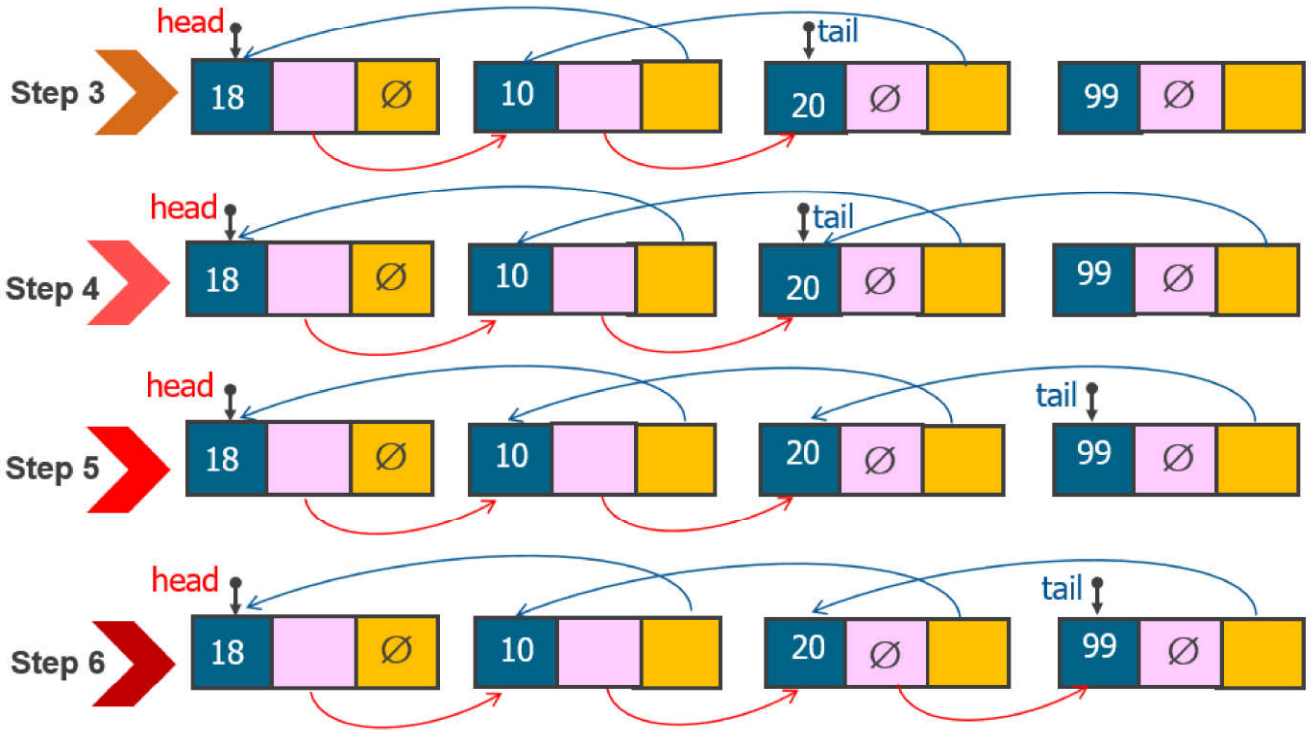
- Inserting node at the end of the doubly linked list
  - ✓Create a new node
  - ✓Initialize the node's `info` member
  - ✓The `next` member is set to `null`
  - ✓The `prev` member is set to the value of `tail`
  - ✓`tail` is set to point to the new node
  - ✓`The next` member of the predecessor is set to point to the new node

Doubly Linked Lists Operations

- Example: insert node 99 at the end of a doubly linked list



Doubly Linked Lists Operations

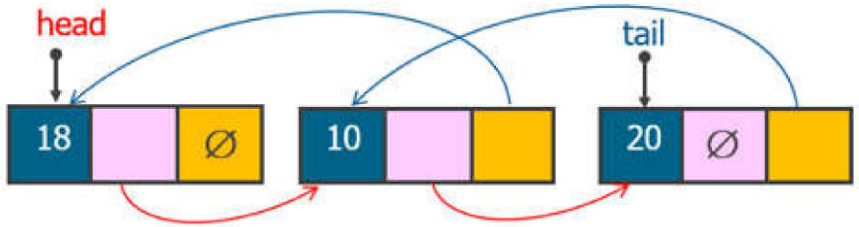


Doubly Linked Lists Operations

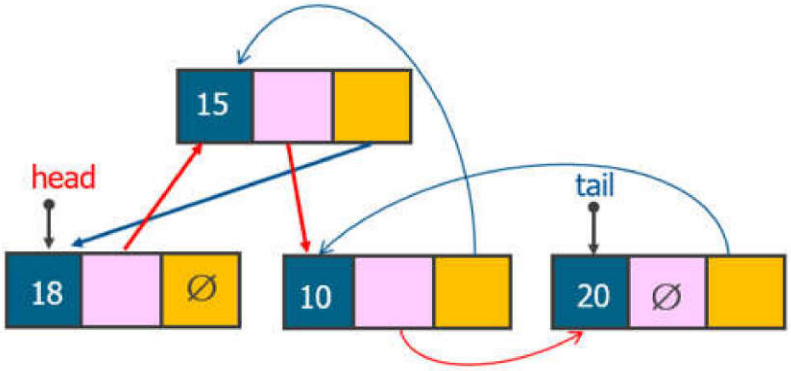
- Inserting node into a doubly linked list
  - ✓Given a node *v* of a doubly linked list, let *z* be a new node that will insert immediately after *v*. Let *w* be the node following *v* (*w* is the node pointed to by *v*'s next link).
  - ✓Operations to insert *z* after *v* into the current doubly linked list:
    - ➢Create a new node (*z*)
    - ➢Initialize the `z's info` member
    - ➢Make `z's prev` link point to *v*
    - ➢Make `z's next` link point to *w*
    - ➢Make `w's prev` link point to *z*
    - ➢Make `v's next` link point to *z*

Doubly Linked Lists Operations

- Example: insert node 15 between node 18 and node 10



- Solution:



Doubly Linked Lists Operations

- Deleting node from the end of the doubly linked list

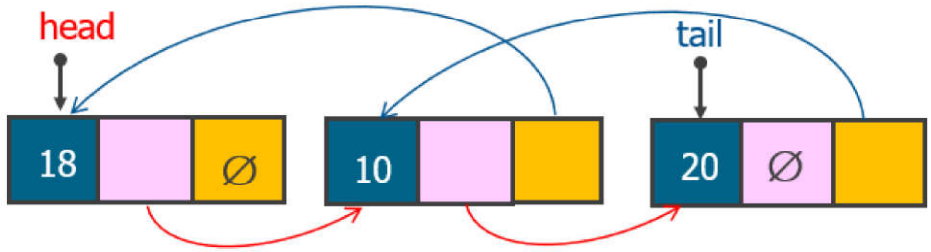
✓`tail` is set to the predecessor of the last node

✓Delete the last node

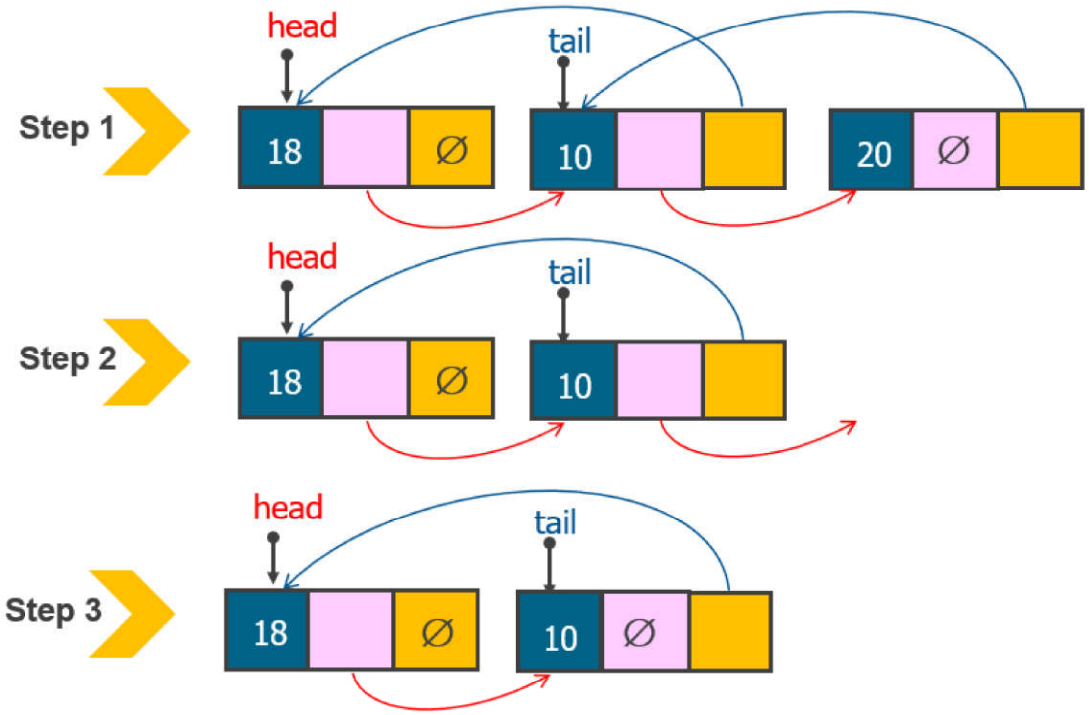
✓The `next` member of the predecessor is set to `null`

Doubly Linked Lists Operations

- Example: delete node 20 from the end of a doubly linked list



Doubly Linked Lists Operations

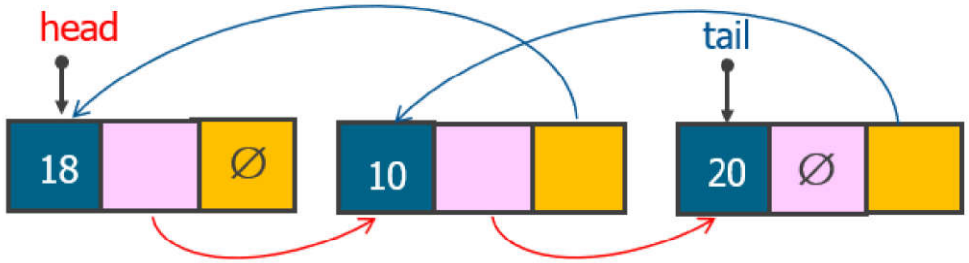


Doubly Linked Lists Operations

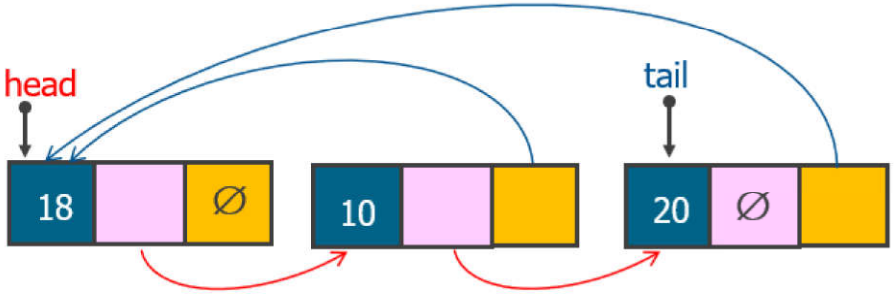
- Deleting node from a doubly linked list
  - ✓Let *v* be a node that will delete, *u* be the node just prior to *v*, and *w* be the node just following *v*.
  - ✓Operations to delete *v* from the current doubly linked list:
    - ➢Make `w's prev` link point to *u*
    - ➢Make `u's next` link point to *w*
    - ➢Delete node `v`

Doubly Linked Lists Operations

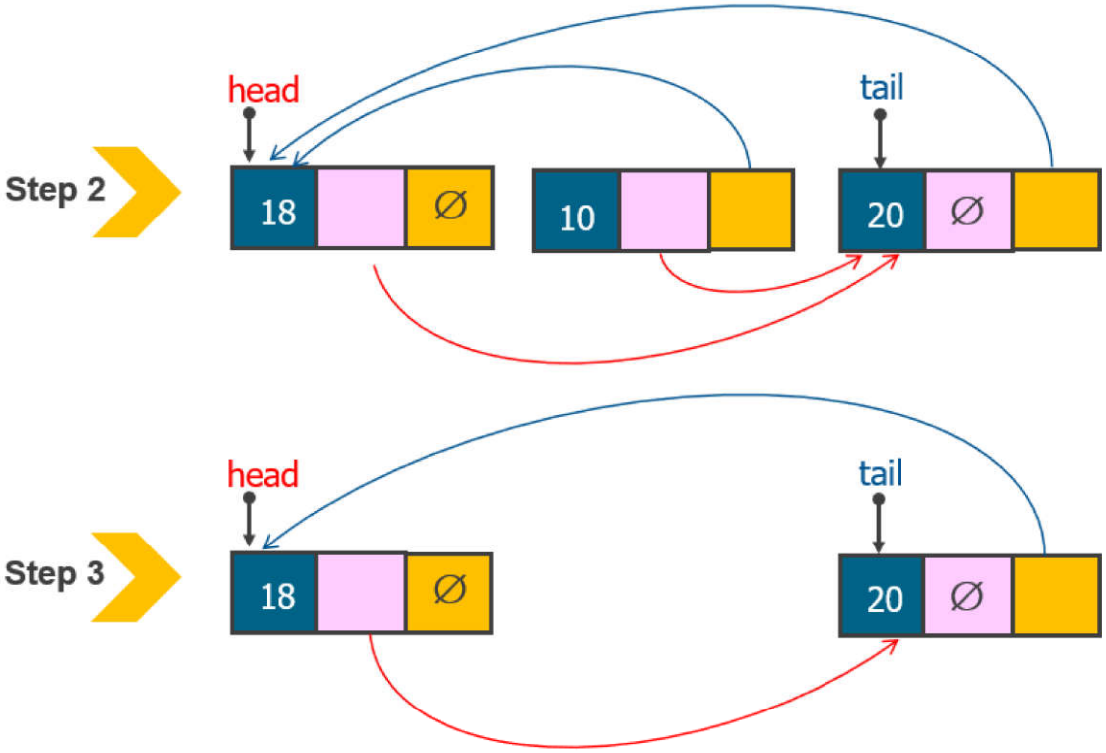
- Example: delete node 10 from a doubly linked list



Step 1

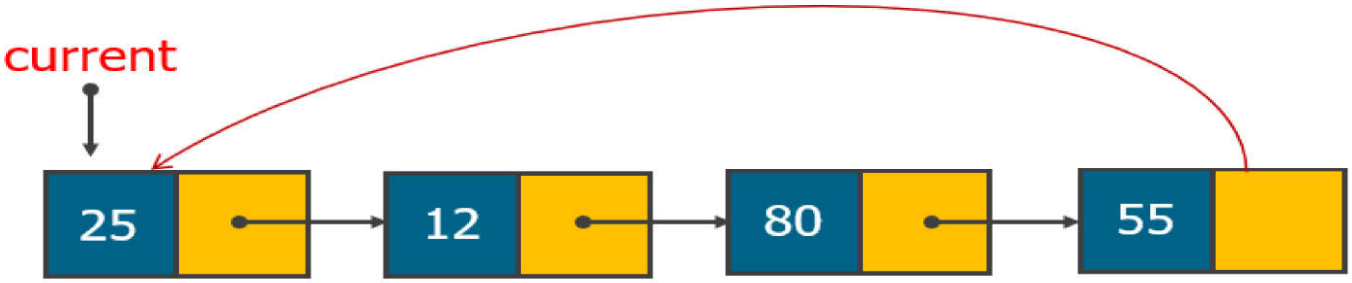


Doubly Linked Lists Operations



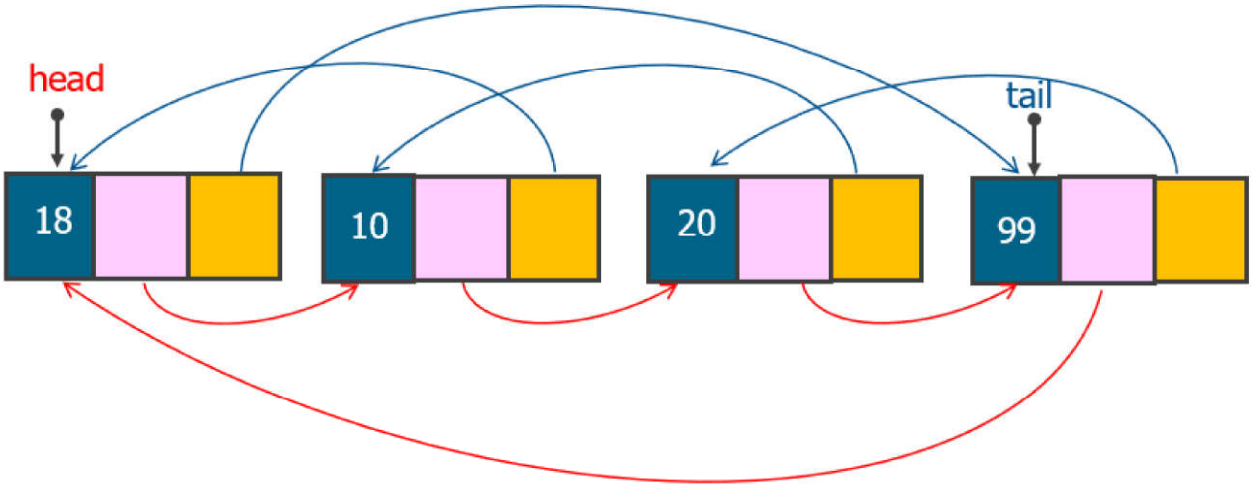
Circular Lists

- Nodes form a ring
- The list is finite and each node has a successor
- `current` is a pointer which uses to access a circular list
- Example: a circular singly linked list



Circular Lists

- Example: a circular doubly linked list



Class Activity

Q & A

Formative Assessment