Data Structures

List Abstract Data Type

- A container of elements that stores each element at a particular position and keeps these positions arranged in a linear order
- It is a general list of the form A0, A1, . . . , Ai-1, Ai, Ai+1, . . ., An-1
- Position is always defined relatively
- Position “p” is always after some position “q” and before some position “s” unless “p” is either first or last position
- Position “p” does not change if we swap or replace the element “e” stored at p with another element
- In the list A0, A1, A2, . . ., An-1, the position of Ai is “i”
- The size is n
- List of size 0 is called an empty list
- Ai follows Ai-1 (i<n) and Ai-1 precedes Ai (i>0)
- A0 is the first element and An-1 is the last element

Lists

- Some of the methods supported by the list ADT are:
- first()
- last()
- isFirst(p)
- isLast(p)
- replaceElement(p, e)
- swapElements(p, q)
- insertFirst(e)
- insertLast(e)
- remove(e)
- removeAtPosition(p)
- Find(e)
- insert(e, p)

Lists: Example

- Example list: 2, 56, 4, 34, 1, 5, 16, 12, 23, 10
- replaceElement(1, 55)
  - 2, 55, 4, 34, 1, 5, 16, 12, 23, 10
- swapElements(2, 3)
  - 2, 56, 34, 4, 1, 5, 16, 12, 23, 10
- insertFirst(22)
  - 22, 2, 56, 4, 34, 1, 5, 16, 12, 23, 10
- insertLast(24)
  - 2, 56, 4, 34, 1, 5, 16, 12, 23, 10, 24
- remove(4)
  - 2, 56, 34, 1, 5, 16, 12, 23, 10
- find(5)
  - Return 5
- Insert(4, 25)
  - 2, 56, 4, 34, 25, 1, 5, 16, 12, 23, 10

Implementation using an array

- A list can be implemented with an N-element array S
- Elements stored from S[0] to S[n-1]; S[0]: the first element and S[n-1] is the last element
- How to insert a new element when the size is N?

Algorithm insert(e, p)

for i=n-1, n-2, . . ., p do

S[i+1] ← S[i] {Make room for “e” to be inserted}

S[p] ← e

n ← n+1

Implementation using an array

Algorithm removeAtPosition(p)

e ← S[p] {e is a temporary variable}

for i=p, p+1, . . ., n-2 do

S[i] = S[i+1] {fill in for the remove element}

n ←n-1

Implementation of Lists

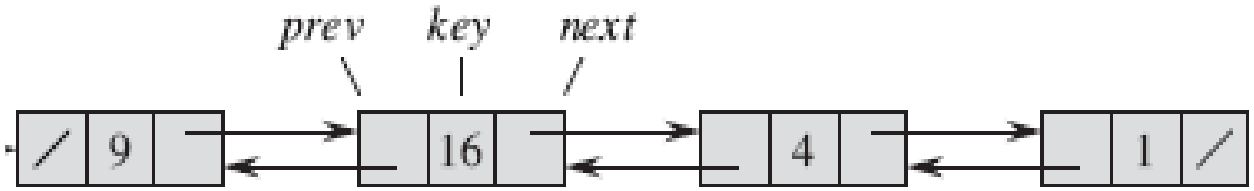
- All operations can be implemented using an array
- Insertion and deletion are expensive, depending upon where the insertions and deletions occur
- Insertion at position “0”
- Deletion of the first element

Implementation of Lists

- A linked list is a data structure where the objects are arranged in a linear order
- In array based implementation, linear order is determined by array indices
- In linked list based implementation, linear order is determined by the pointer or link maintained in each object

Doubly linked list

- This is an example of a doubly linked list



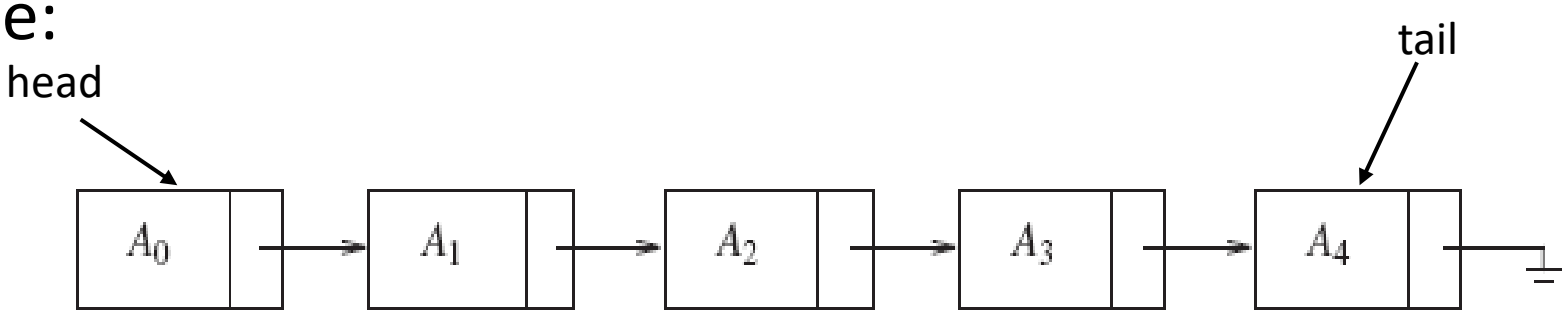
- Each node/object of a doubly linked list stores a “key/element” and two links/references called “next” and “prev”
- The “next” link points to the next node in the list and the “prev” link points to the previous node in the list
- Can be traversed in either direction
- The “next” and “prev” links of a node
- If the “next” or “prev” links of a node are “NIL”, then?

Variants of a linked list

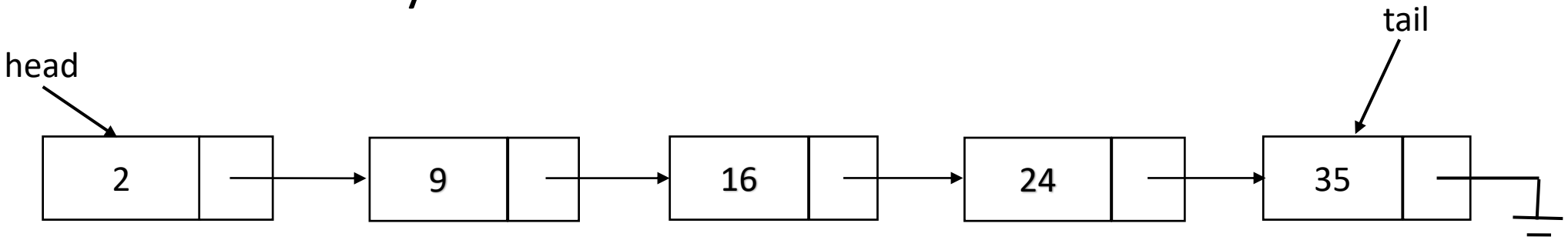
- A list may have one of the several forms:
  - Singly linked or doubly linked
  - Sorted or not
  - Circular or not
  - Lists with sentinels

Singly linked list

- If a list is singly linked, then each node have a link/reference to only the next node in the list
- Example:

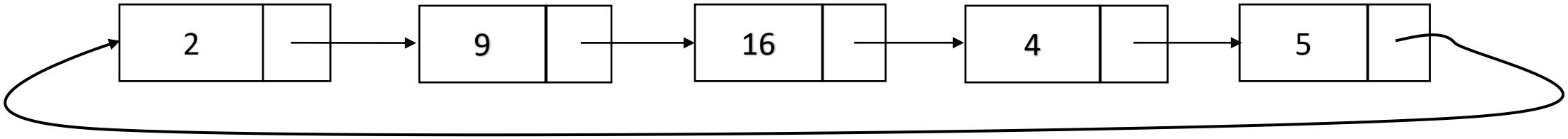


- If a list is sorted, the linear order of the list corresponds to the sorted order of the keys stored in nodes

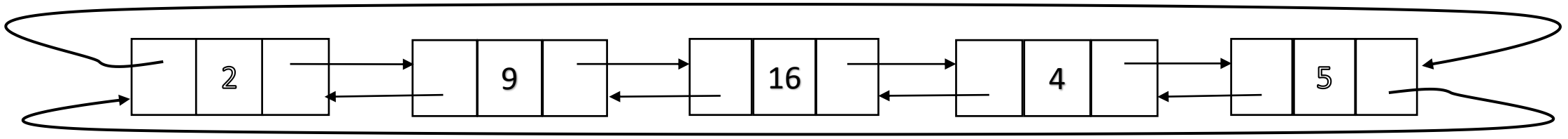


Circular singly linked list

- If the next link of the last node points to the first node, then the resultant list is called a circular singly linked list



- Circular doubly linked list: Make the “next” link of the tail to point to the head and the “prev” link of the head to point to the tail

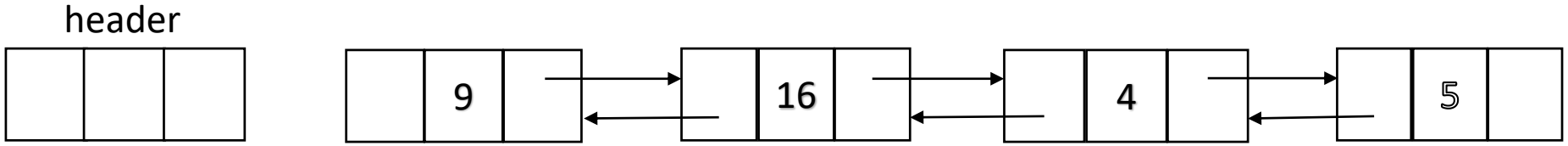


Linked lists with sentinels

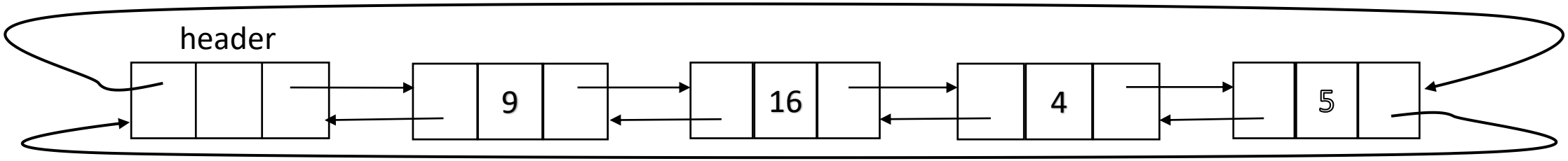
- To simplify search and update operations, it is convenient to add a special node at the beginning/ending of a list: a header node just before the head of the list; a trailer node after the tail
- A sentinel is a dummy node does not store any data element but has all valid links similar the other nodes

Linked lists with sentinels

- Consider a doubly linked list



- The “next” link of header points to the head of the list and the “prev” link points to the tail of the list
- Also make the “next” link of the tail to point to the header and the “prev” link of the head to point to the header
- The resultant list is called “circular doubly linked list with sentinel”



Implementation using linked lists

Algorithm insertAfter(p, e)

create a new node v

v.element = e

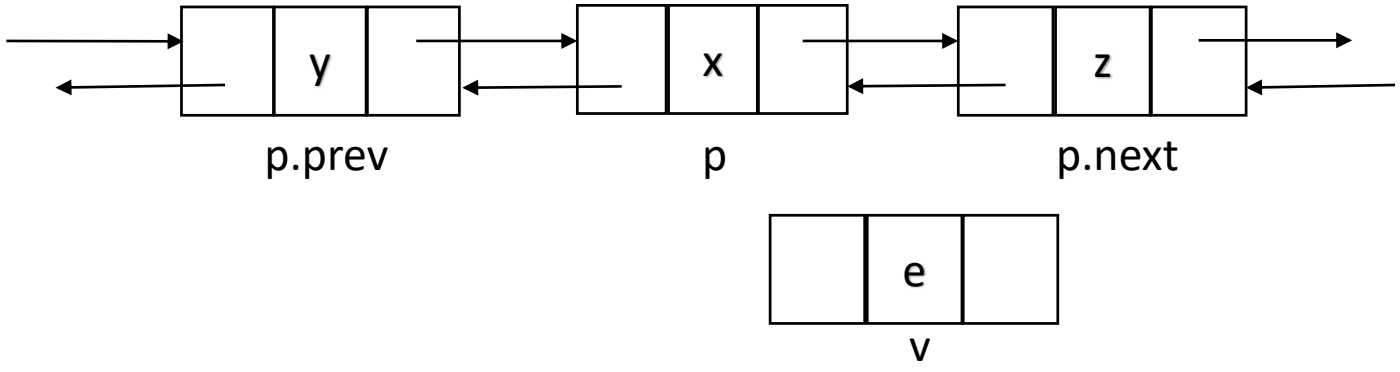
v.prev = p {link v to its predecessor}

v.next = p.next {link v to its successor}

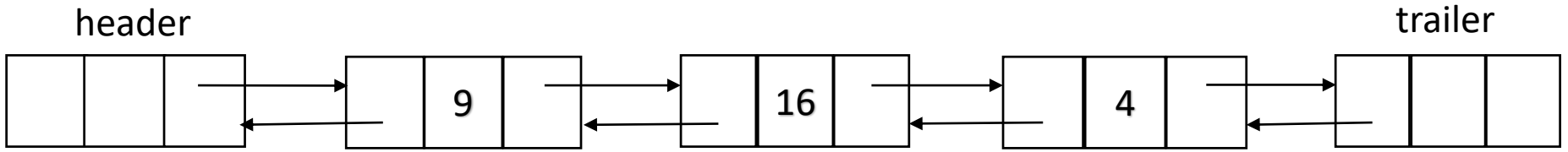
(p.next).prev = v {link p's old successor to v}

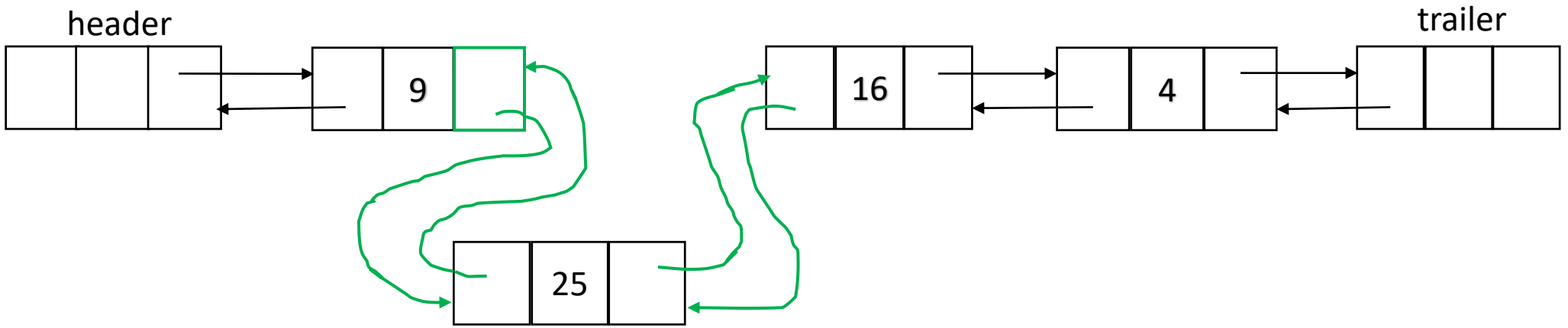
p.next = v {link p's old predecessor to v}

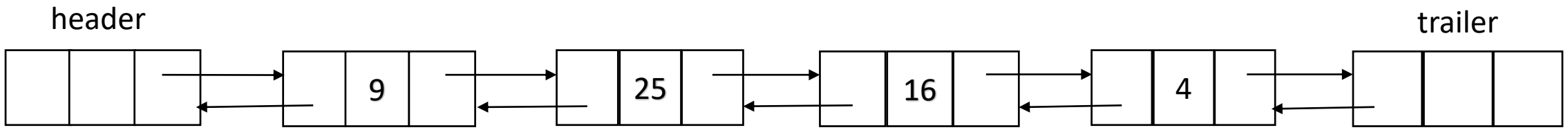
return v {the position of element "e"}



Insertion: Example







Implementation using linked lists

Algorithm removeAtPosition(p)

t ← p.element {a temporary variable to hold the return value}

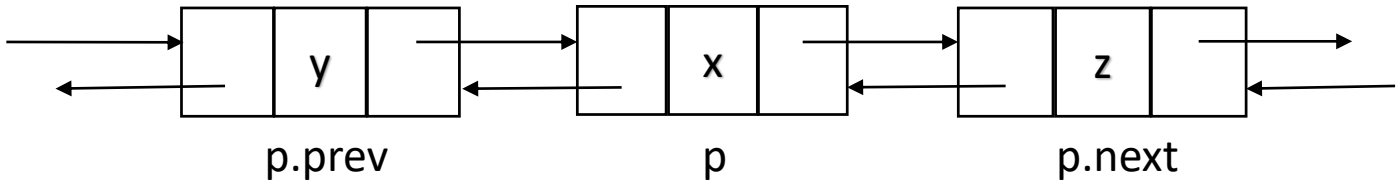
(p.prev).next = p.next {link out p}

(p.next).prev = p.prev {link out p}

p.prev = null {invalidate the position p}

p.next = null

return t



Deletion: Example