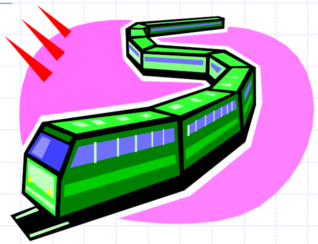
Lists



Position ADT

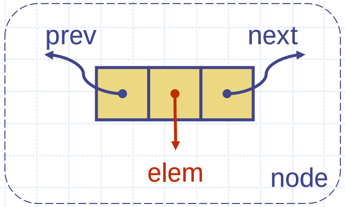
- The Position ADT models the notion of place within a data structure where a single object is stored
- It gives a unified view of diverse ways of storing data, such as
  - a cell of an array
  - a node of a linked list
- Just one method:
  - object p.element(): returns the element at position
  - In C++ it is convenient to implement this as *p

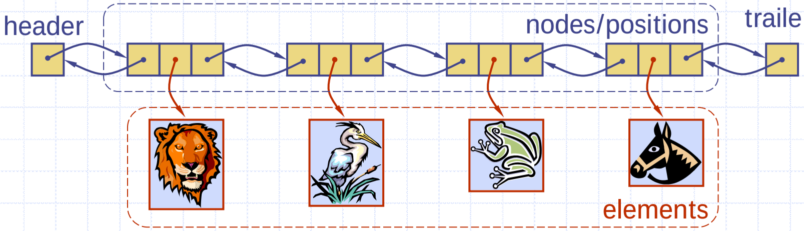
Node List ADT

- The Node List ADT models a sequence of positions storing arbitrary objects
- It establishes a before/after relation between positions
- Generic methods:
  - size(), empty()
- Iterators:
  - begin(), end()
- Update methods:
  - insertFront(e), insertBack(e)
  - removeFront(), removeBack()
- Iterator-based update:
  - insert(p, e)
  - remove(p)

Doubly Linked List

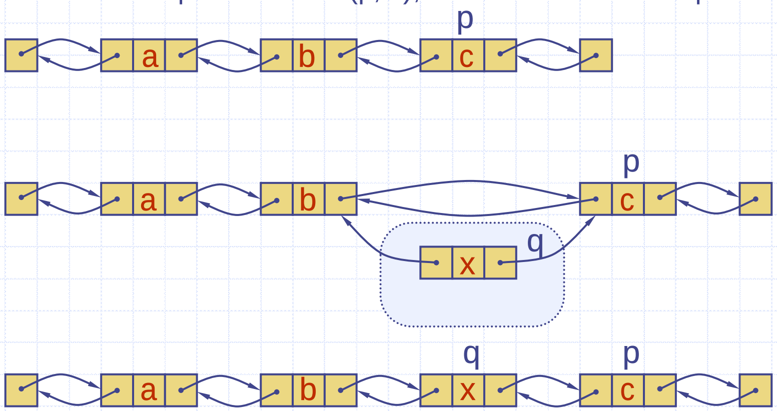
- A doubly linked list provides a natural implementation of the Node List ADT
- Nodes implement Position and store:
  - element
  - link to the previous node
  - link to the next node
- Special trailer and header nodes





Insertion

- We visualize operation insert(p, x), which inserts x before p

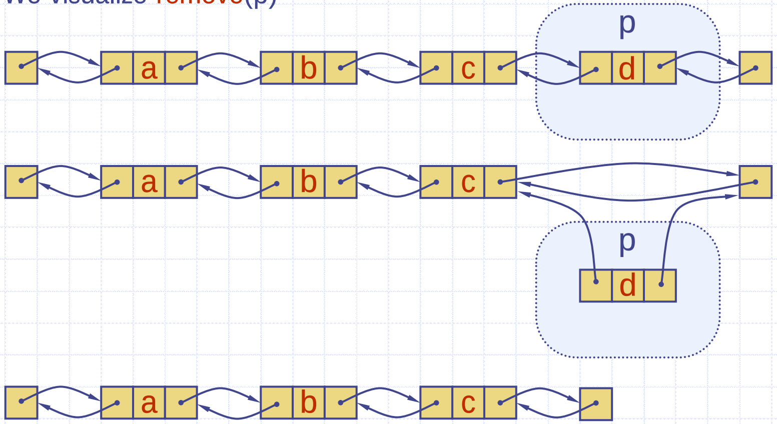


Insertion Algorithm

```
Algorithm insert(p, e): {insert e before p}
    Create a new node v
    v→element = e
    u = p→prev
    v→next = p;  p→prev = v  {link in v before p}
    v→prev = u;  u→next = v  {link in v after u}
```

Deletion

- We visualize remove(p)



Deletion Algorithm

```
Algorithm remove(p):
    u = p→prev
    w = p→next
    u→next = w {linking out p}
    w→prev = u
```

Performance

- In the implementation of the List ADT by means of a doubly linked list
  - The space used by a list with n elements is $O(n)$
  - The space used by each position of the list is $O(1)$
  - All the operations of the List ADT run in $O(1)$ time
  - Operation element() of the Position ADT runs in $O(1)$ time