Algorithms and Data Structures (CSci 115)

California State University Fresno

College of Science and Mathematics

Department of Computer Science

H. Cecotti

Warning !

- For labs and projects
  - You can easily find the solutions of the labs and the content of the methods on the internet
    - State of the art exercises
  - If you do everything with just copy/paste…
    - Be careful to what will happen during the midterms and the final
      - → No documents allowed → No copy/paste from existing code !!!
    - Copy/paste only, even if you understand → no work from your memory
  - What to do:
    - Get the principle
    - Code from scratch
      - Be as clean and rigorous as possible
      - If you don't understand **exactly** what you re writing, stop and use paper and pen until you are sure about what you need to implement on the computer
    - Test with examples to see if it works

Learning outcomes

- Data structures
  - Double linked lists
  - Circular lists

Motivations

- Development of an application
  - Need of a data structure
    - Size: fixed or variable
    - Type of operations
      - Insertion
      - Deletion
      - Search
  - Ideal data structure
    - all the operations are in O(1)

Double-linked lists

- A double-ended list is similar to an ordinary linked list, but it has one additional feature: a reference to the last link as well as to the first – often referred to as the **tail**

- The reference to the last link permits you to insert a new link **directly** at the end of the list
  - No need to iterate through the entire list until you reach the end

- Access to the end of the list as well as the beginning makes the double-ended list
  - suitable for certain situations that a single-ended list can’t handle efficiently
  - Like the Queue

Double-linked lists

- Definition
  - Node with 3 elements
    - Data
    - Pointer to Next element
      - Null if last element of the list
    - Pointer to Previous element
      - Null if first element of the list

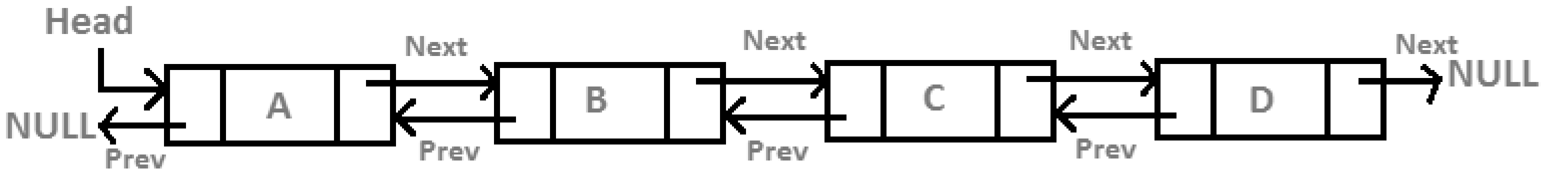
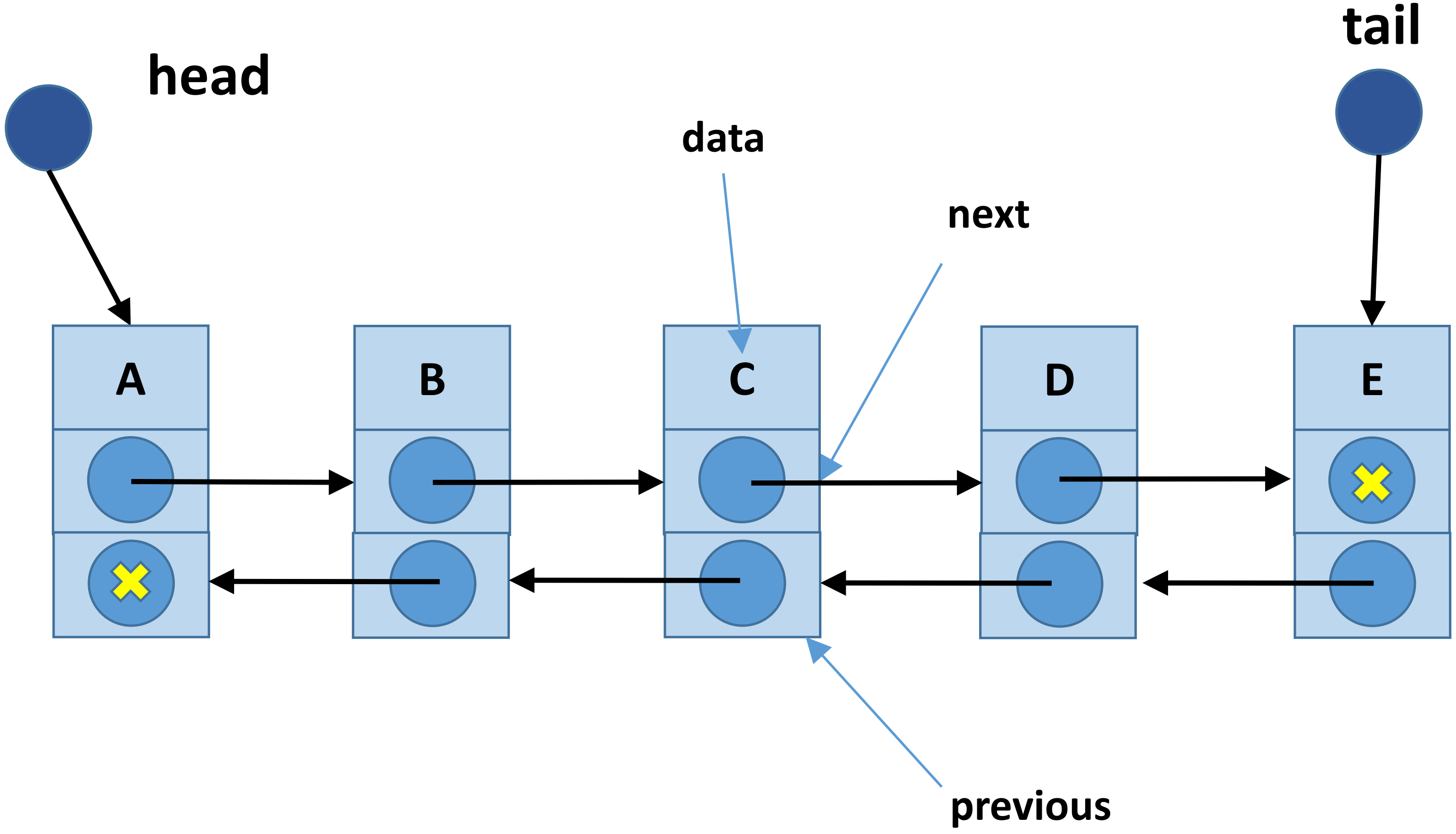


Diagram of a Double-Linked List

-

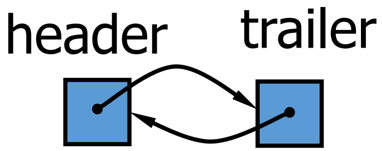


Functions

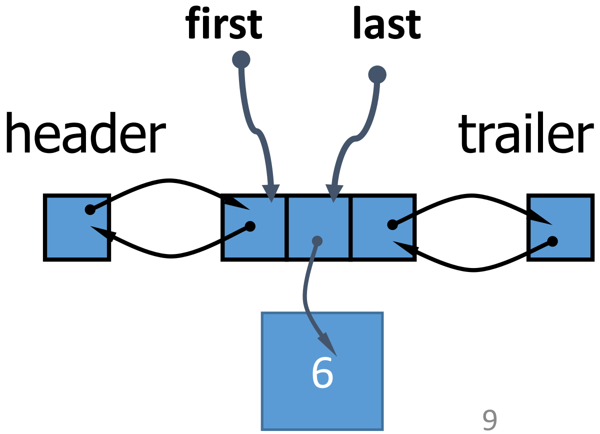
- Insert
  - Head
  - Tail
  - Middle
    - After a particular element
- Delete
  - Head
  - Tail
  - Middle
    - After a particular element
- Search
  - Return the element with a particular value
- Display
  - Display all the elements in the list

Double-linked lists

- What we need in the class
  - Reference to sentinel head-node
  - Reference to sentinel tail-node
  - Size-counter that keeps track of the number of nodes in the list
    - excluding the 2 sentinels



- Special case
  - Empty list
    - Size = 0
    - head.next = tail
    - tail.prev = head
  - Single Node List:
    - Size = 1
      - first node = last node
    - first node: head.next
    - last node: tail.prev



Pseudo code

- Add a new element at the beginning of the list
  - Algorithm addFirst()

```
new(T)
T.data ← y
T.next ← head.next
T.prev ← head
head.next.prev ← T
head.next ← T
Size++
```

Pseudo code

- Add a new element at the end of the list
  - Algorithm addLast()

```
new(T)
T.data ← y
T.next ← tail
T.prev ← tail.prev
tail.prev.next ← T
tail.prev ← T
Size++
```

Pseudo code

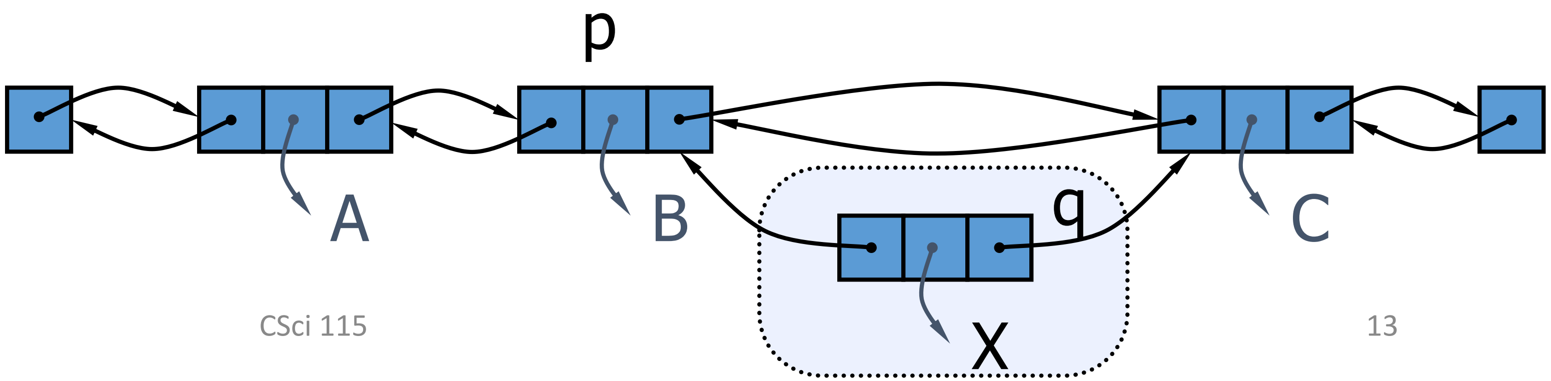
- Remove last
    - Warning: before removal → check for empty list
        - If not empty, remove the last node in the list
- Algorithm to remove the last element

```
T ← tail.prev
y ← T.data
T.prev.next ← tail
tail.prev ← T.prev
delete(T)
size--
return y
```

Pseudo code

- Insertion
  - Algorithm insertAfter(p,e):

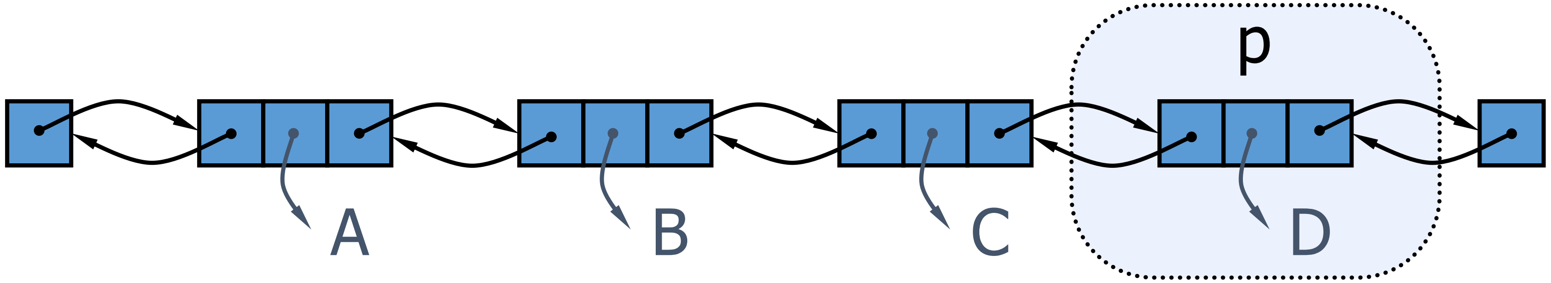
```
○ Create a new node v
○ v.setElement(e)
○ v.setPrev(p)            // link v to its predecessor
○ v.setNext(p.getNext())  // link v to its successor
○ (p.getNext()).setPrev(v) // link p's old successor to v
○ p.setNext(v)            // link p to its new successor, v
○ return v                // the position for the element e
```



Pseudo code

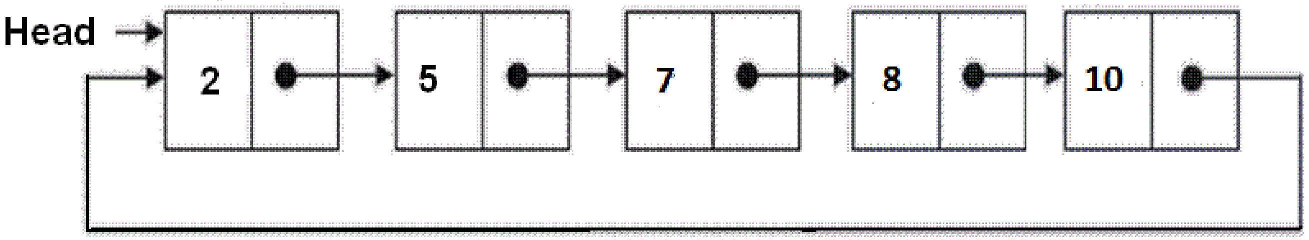
- Remove a node
  - Algorithm remove(p):

```
o t = p.element    // tmp variable to hold the return value
o (p.getPrev()).setNext(p.getNext())
o (p.getNext()).setPrev(p.getPrev())
o p.setPrev(null)
o p.setNext(null)
o return t
```



Circular lists

- In linear linked lists
  - if a list is traversed (all the elements visited) an external pointer to the list must be preserved in order to be able to reference the list again.
- Circular linked lists
  - used to help the traverse the same list again and again if needed.
  - similar to the linear list where in the circular list the pointer of the last node points not NULL but the first node.
- Goal
  - There is always a next element



Circular lists

- Definition
  - Node with 2 elements
    - Data
    - Pointer to the Next element
      - Simple chained list: Next of last element = NULL
      - Circular list: Next of the last element → First element = Head
  - 2 methods to know if a node is the first node or not
    - a external pointer, list, points the first node a header node is placed as the first node of the circular list.
      - can be separated from the others by
        - a sentinel value as the info part
        - a dedicated flag variable to specify if the node is a header node or not.

Circular lists

- With header node
  - The header node in a circular list can be specified by a sentinel value or a dedicated flag:
  - Header Node with Sentinel
    - We consider that info part contains positive integers (>0)
    - → the info part of a header node can be -1.
  - Example for a sentinel used to represent the header node:

```
struct node{
        int info;
        struct node *next;
}; typedef struct node *NODEPTR;
```

Circular lists

- Header Node with Flag
  - a extra variable **flag**
    - used to represent the header node.
  - For example
    - flag in the header node can be 1, where the flag is 0 for the other nodes.

```
struct node{
        int flag;
        int info;
        struct node *next;
}; typedef struct node *NODEPTR;
```

Conclusion

- Data structures
  - Double chained lists
    - Going both ways → quick access to previous and next element
  - Simple chained Circular list
    - You may implement the double chained list
- Type of lists
  - Depends on the problem
- Question
  - We would like a “list” with access in O(1)
    - What to do?

Questions ?

- Reading:
  - ➢CSci 115 book: Section 5.2 (double chained list)
  - ➢Csci 115 book: Section 5.3 (circular list)