Data Structures

Linked Lists

CS 225
Brad Solomon & G Carl Evans

August 30, 2023

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

ACM Fall Open House

Tues, September 5th, 6:30–9:00 pm
CIF RM 0035 and 0027

Join us at our annual Open House and learn about ACM, other amazing RSOs in the CS department, and how to get involved!

Dinner will be provided!

Exam 0 (August 29 — 31)

An introduction to CBTF exam environment / expectations

Quiz on foundational knowledge from all pre-reqs

Practice questions can be found on PL

Topics covered can be found on website

If you haven’t yet signed up do so ASAP!

Exam 1 (September 11 — 13)

7

A mixture of multiple choice** and coding questions

Exam on content up to **September ~~4~~th** 1

Prairielearn will have a practice exam sometime next week

Sign up also began August 24th

* - may not all be MC

* - small amount of MC

MP_stickers (Due September 11th)

An introductory assignment

Consider the Rule of Three (Rule of Zero)

Good practice on defining classes

A lot of the functions here are simple — don’t share code!

Make sure you understand **PNG** and **HSLAPixel**

Be Respectful: Noise Levels

Class runs from 11:00 — 11:50 AM

The last minutes of lecture normally wraps up a point, opens the floor for questions, or asks you to think critically about a topic we will start the following class. **All of this is important!**

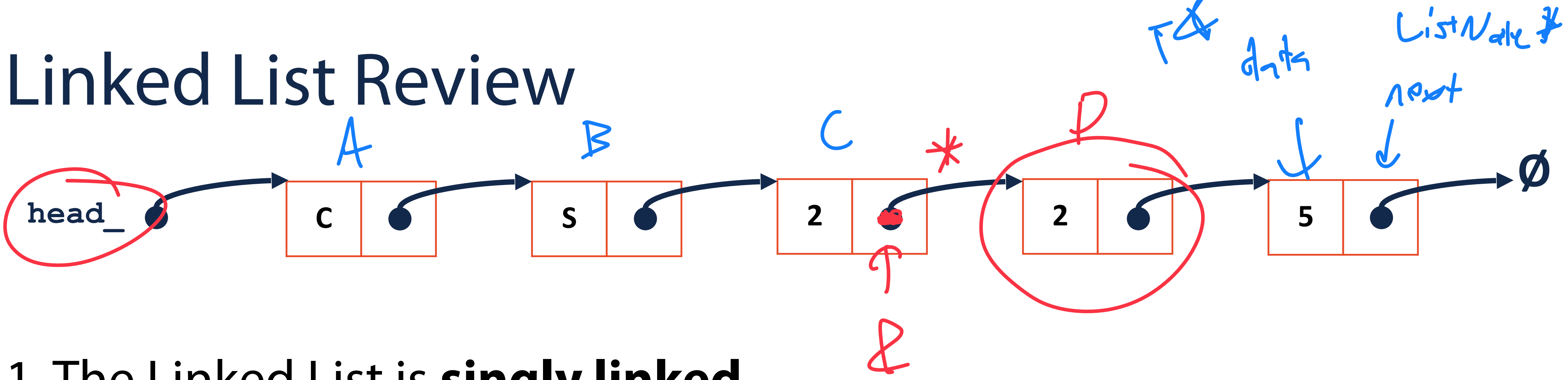
Please don’t start to leave until class has wrapped up for the day

Learning Objectives

Review linked list operations (and go over new ones)

Introduce array list implementations

Linked List Review



1. The Linked List is **singly linked**

↳ can only move in one direction

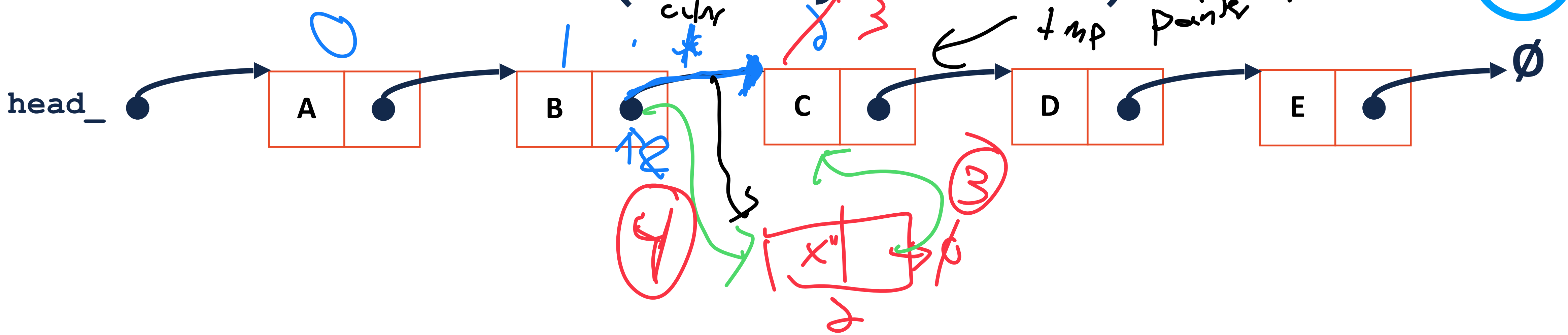
B is A->next / C is A->next->next

2. The Linked List does not permit **random access** A->B->C

↳ only ListNode store in List class is head

3. _index(index) returns a **reference to a pointer**

Linked List: insert(data, index)



1) Create a new nod

2) Get ref to pointer that points to current node @ index (_ intexo/index)

3) Link my new node into chain

↳ 3) Set our new nodes next to be pointed at existing node (at index)

4) Set curr equal to new node

List Random Access [ ]

L = [A, B, C, D]

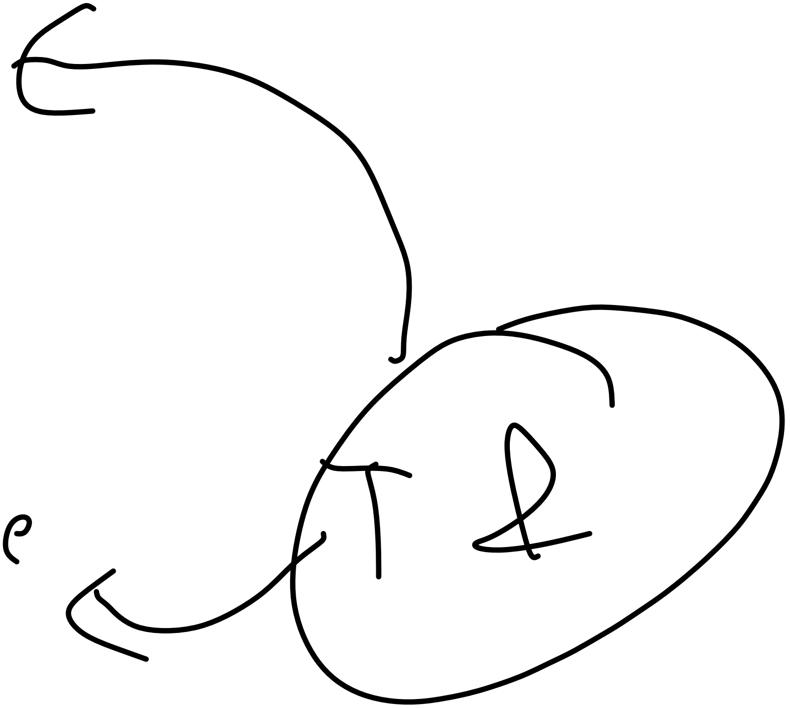
Given a list L, what operations can we do on L[ ]?

L[1]

print L[1] # get the value

remove (L[1])

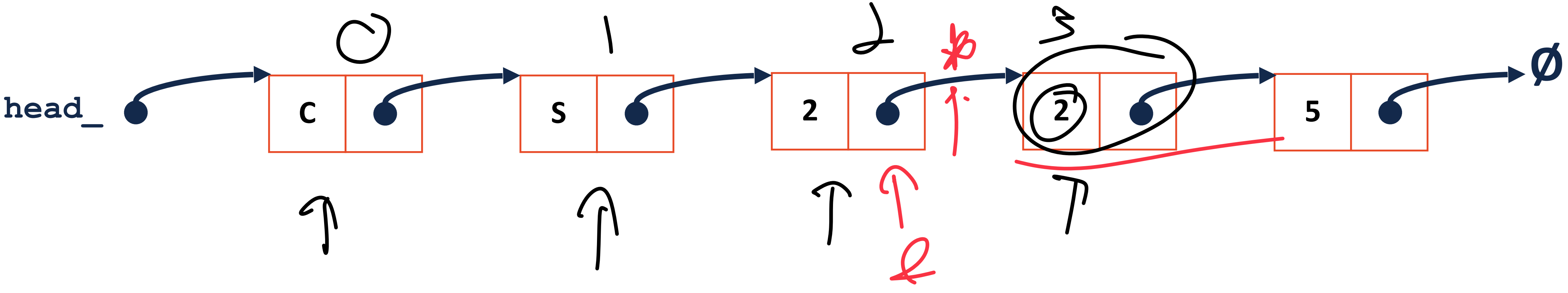
L[1] = E # set the value



3

List.hpp

```
template <typename T>
T & List<T>::operator[](unsigned index) {

    ListNode *& curr = _index(index);


    return curr->data;



}
```

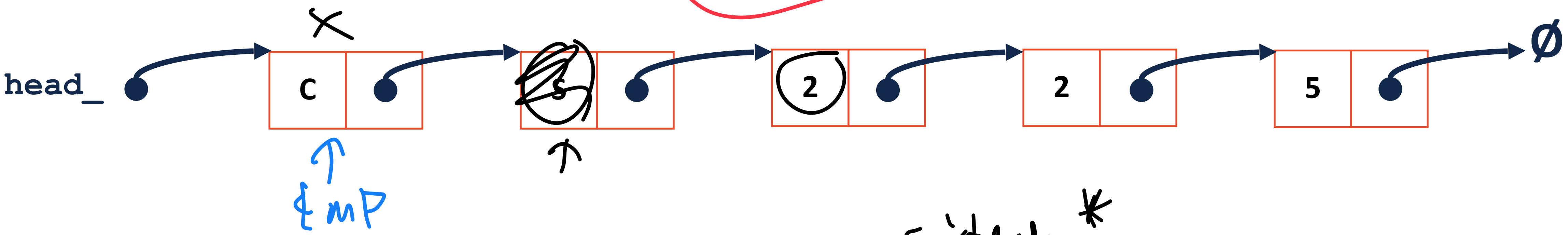


query

Linked List: find(~~data~~)

not index

data = "2"



ListNode *

this sort of loops!

1) Make a tmp node equal to head

2) check if tmp has data equal to query

3) Set tmp = tmp -> next

4) Repeat (2) & (3) until found or end of list

5) return tmp

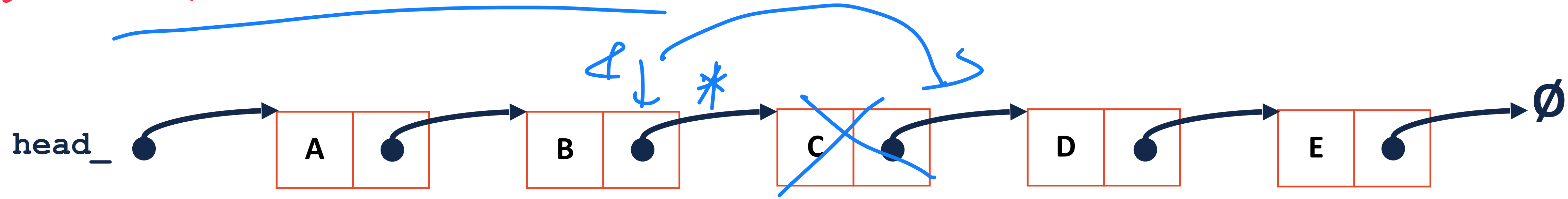
Linked List: Remove(<parameters>)

What input parameters make sense for remove?

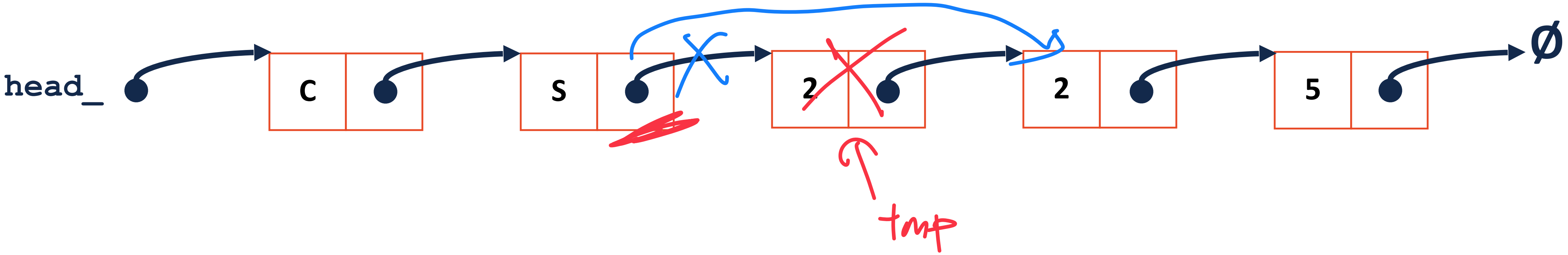
remove (unsigned index) ← index(index) simplifies problem

remove (T & value) ← find(value) simplifies problem

remove (ListNode *& tmp) ← is easy and fast



Linked List: remove(ListNode *& node)



1) Need to make a tmp pointer

2) node = node→next; // node = tmp→next;

3) delete tmp;

O(1)

You decide!

List.hpp

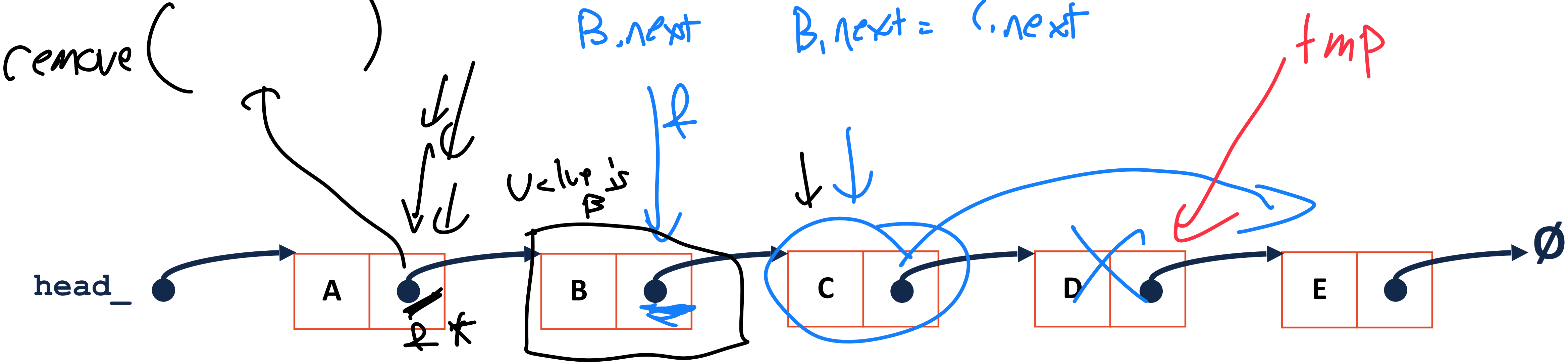
```
template <typename T>
T List<T>::remove(ListNode *& node) {

ListNode * tmp = node;

node = node->next;

T data = tmp->data;

delete tmp;

return data;

}
```

_head = nullptr

B -> next is C

// removal

So I can return data



Linked List: `remove`

What is the running time to remove (if given a reference to a pointer)?

↳ O(1)

index

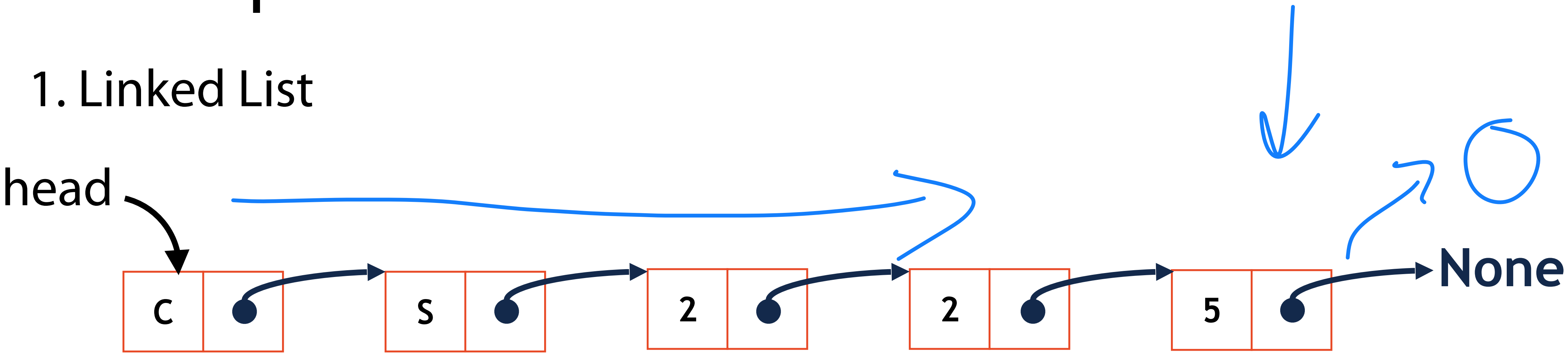
What is the running time to remove (if given a value)?

↳ O(n) because I do find (O(n))

O(n) b/c _index is O(n)

List Implementations

1. Linked List



Pro

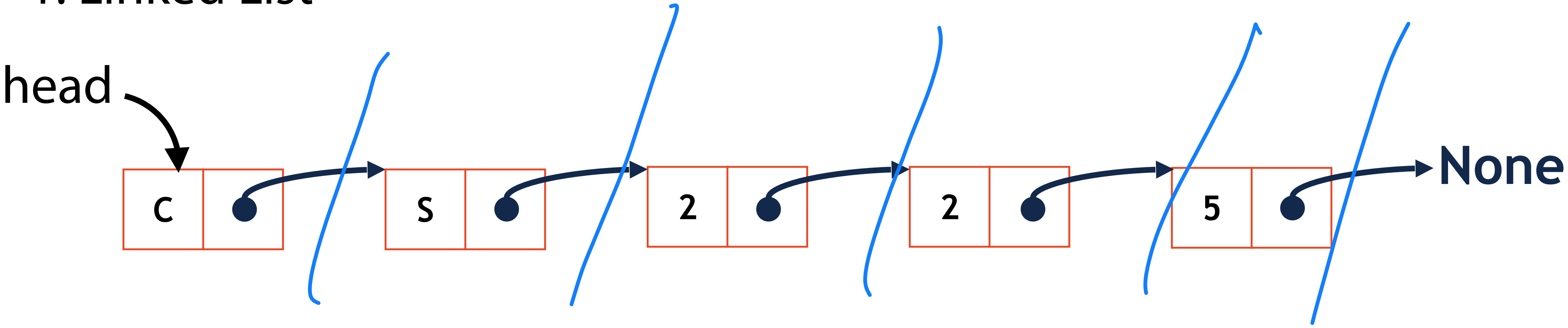
1) We can change LL easily *
* if we have address

Con

2) It is hard to get these addresses

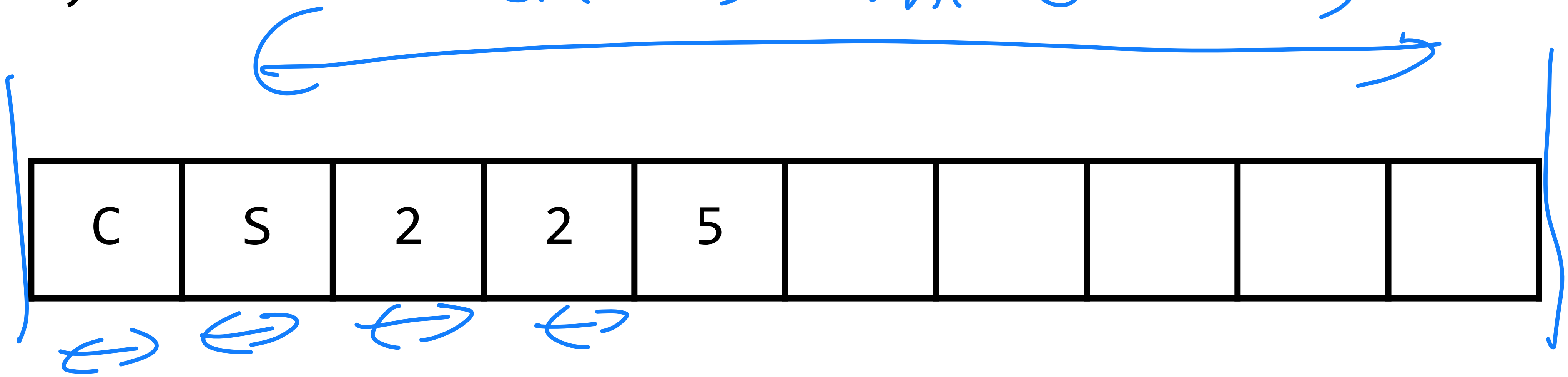
List Implementations

1. Linked List



2. Array List

One big chunk of memory



List ADT

→ cc handles all of

→ array list?

1. Insert

Are there good places to insert into an array?

2. Delete

good places to delete? good parameters

3. isEmpty

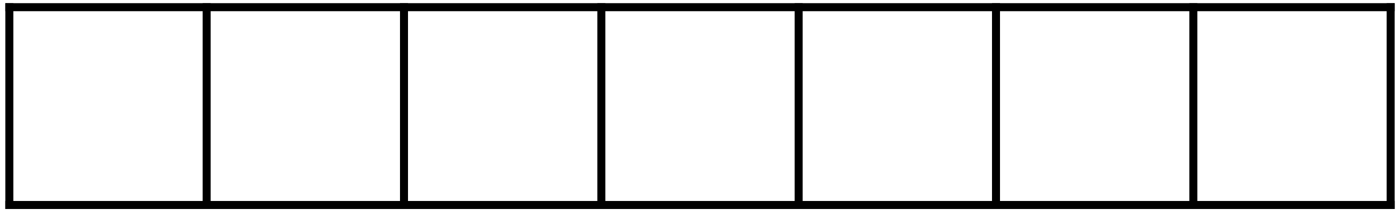
Are these fast? Easy?

4. getData

5. Create an empty list

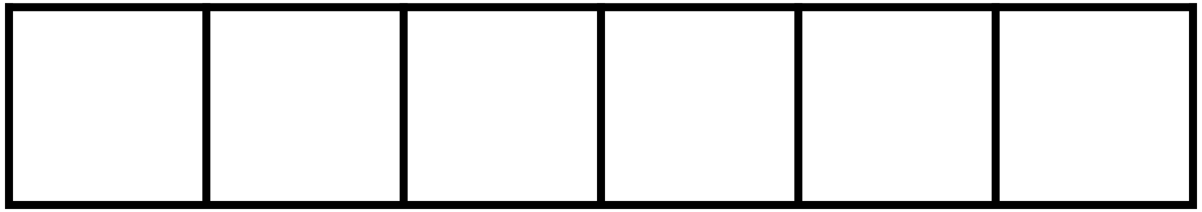
What information do I need to know?

Array List



List.h

```
#pragma once

template <typename T>
class List {
public:
    /* --- */
private:
  T *data_;

  T *size;

  T *capacity;

    /* --- */
};
```



Array List: [ ]

C	S	2	2	5					

Array List: `insertAtFront(data)`

C	S	2	2	5					