Data Structures
Linked Lists

CS 225
Brad Solomon & G Carl Evans

August 28, 2023

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

EXPERT SPEAKERS

TECH TALKS

PUZZLEBANG

CAREER FAIR

HACKATHON

FREE FOOD + SWAG

SEPT 18 - 22

SCAN ME

reflections | projections

Discord Question Helpers

Glad to see so many people using Discord in lecture

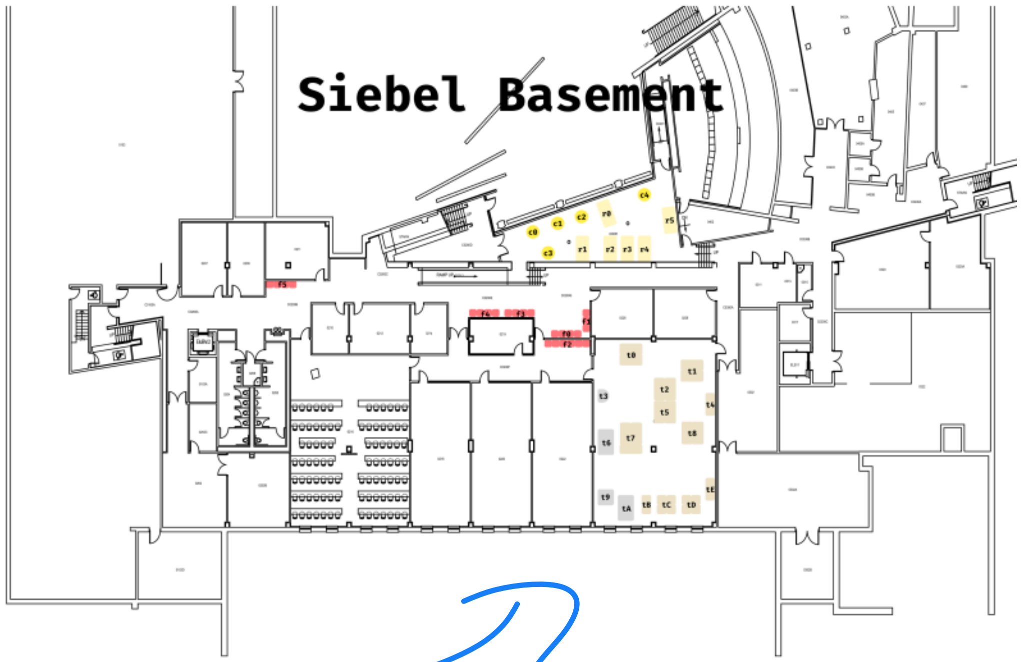
To help answer questions in class, we will have staff members monitoring Discord.

Office Hour Etiquette

Schedule and link to queue on the website

Pay attention to the rules!

1. Be in Siebel Basement
2. Tag questions #mp1 #potd
3. Ask **one** specific question
4. Include a specific location
5. Include both your name and Discord ID



Learning Objectives

Review linked list operations (and go over new ones)

Introduce array list implementations

List.h

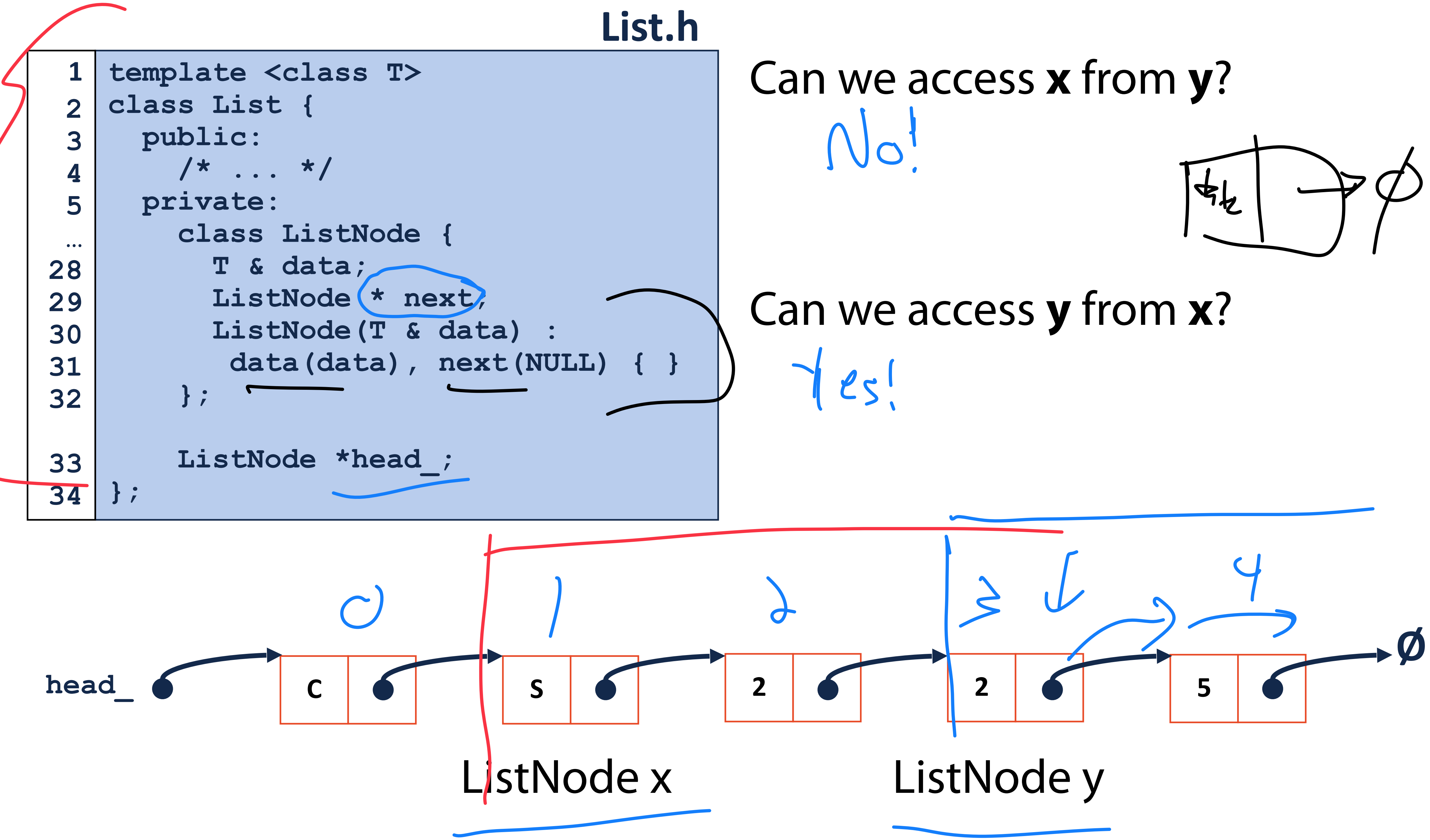
```
template <class T>
class List {
  public:
    /* ... */
  private:
    class ListNode {
...
      T & data;
      ListNode * next;
      ListNode(T & data) :
       data(data), next(NULL) { }
    };

    ListNode *head_;
};
```

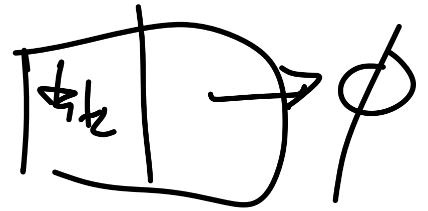
Can we access **x** from **y**?

No!

Can we access **y** from **x**?

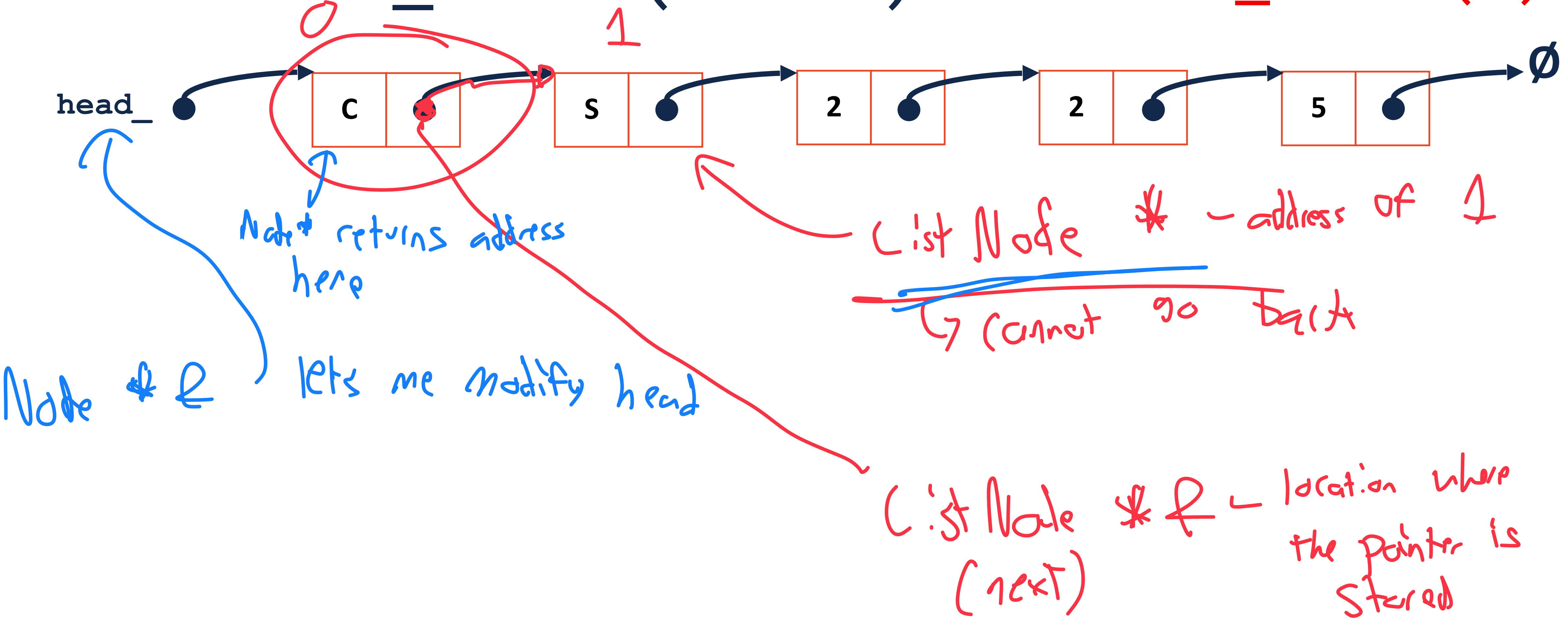
Yes!





Linked List: _index(index)

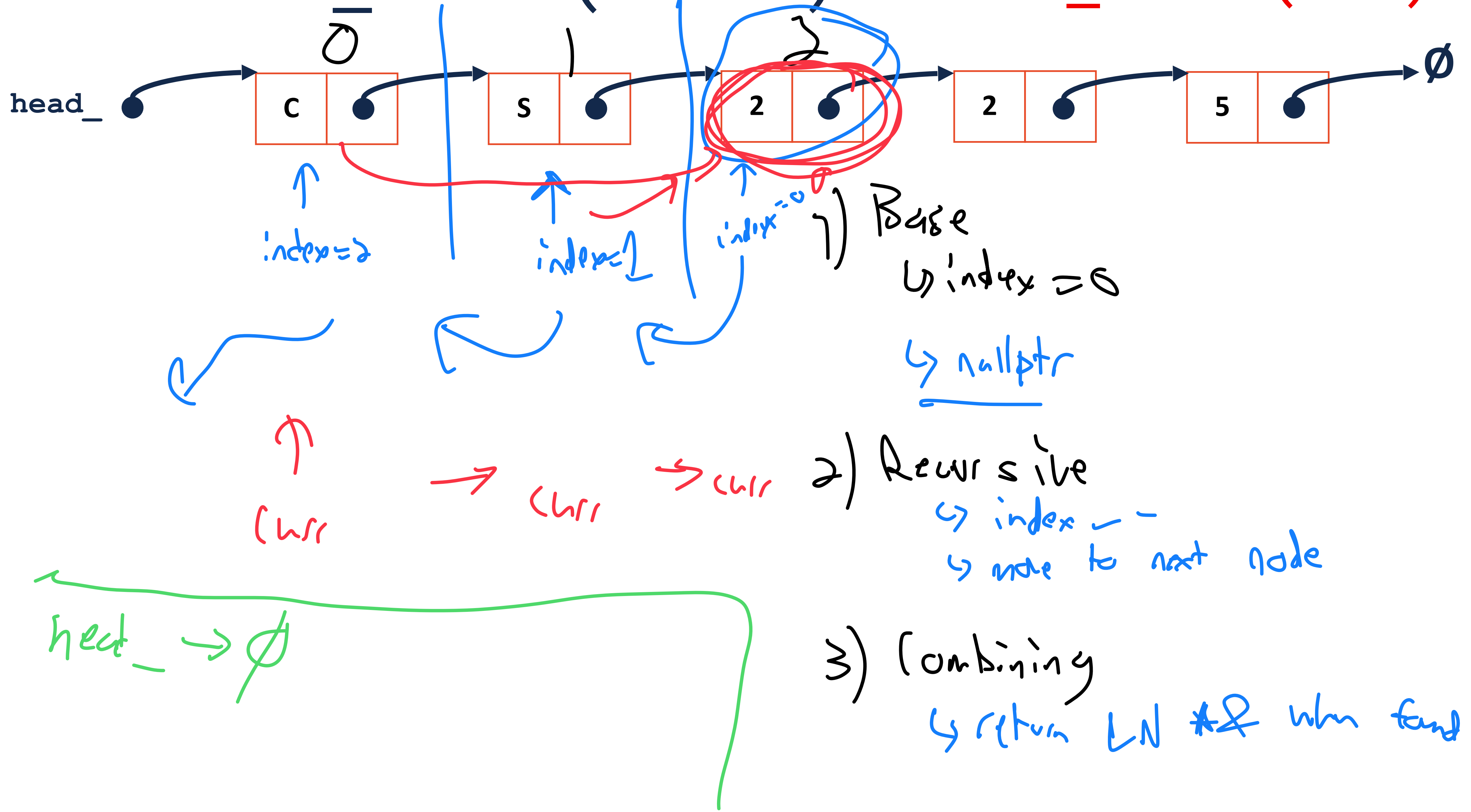
_index(0)



head_
C
S
2
2
5
Ø
0
1
Node* returns address here
Node *& lets me modify head
ListNode * ← address of 1
~~↳ cannot go back~~
ListNode *& ← location where the pointer is stored
(next)

Linked List: `_index(index)` ListNode *&

`_index( 2 )`



1) Base
   ↳ index = 0
   ↳ nullptr

2) Recursive
   ↳ index --
   ↳ move to next node

3) Combining
   ↳ return LN *& when found

List.hpp

interface

```
template <typename T>
typename List<T>::ListNode *& List<T>::_index(unsigned index){
        return _index(index, head_)
}
```

2 0 0 0

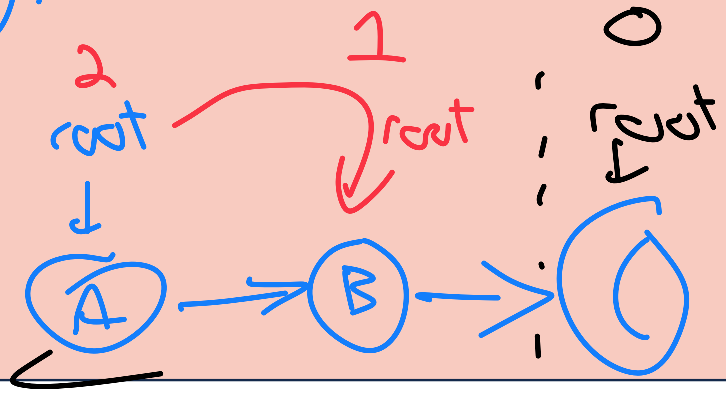
implementation

```
template <typename T>
typename List<T>::ListNode *& List<T>::_index(unsigned index, ListNode *& root){
    if(index == 0){ return root;}

    if(root == nullptr){ return root;}


    return _index(index-1, root->next);
}
```

2

head_

nullptr->next



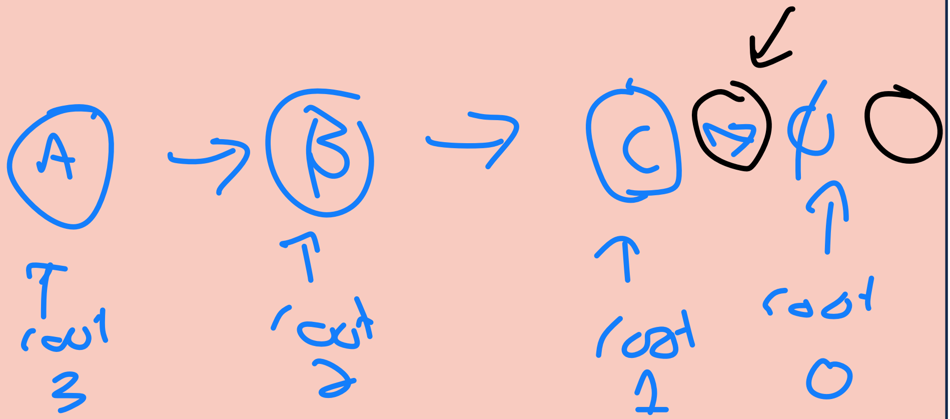
List.hpp

```
template <typename T>
typename List<T>::ListNode *& List<T>::_index(unsigned index){
        return _index(index, head_)
}
```

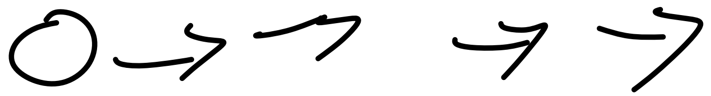
```
template <typename T>
typename List<T>::ListNode *& List<T>::_index(unsigned index, ListNode *& root){


    if (index == 0){ return root; }


    if (root == nullptr){ return root; }


    return _index(index - 1, root -> next);


}
```

3

head



root -> next = new LN



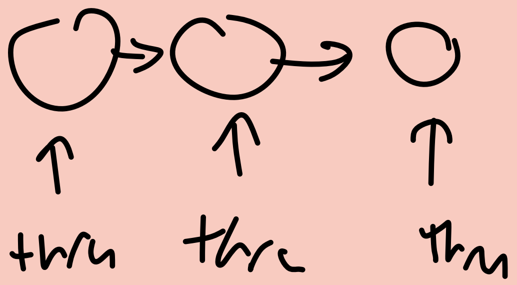
$n = 2 * 10^{9999}$

List.hpp

```
// Iterative Solution:
template <typename T>
typename List<T>::ListNode *& List<T>::_index(unsigned index) {
  if (index == 0) { return head; }
  else {
    ListNode *thru = head;
    for (unsigned i = 0; i < index - 1; i++) {
      thru = thru->next;
    }
    return thru->next;
  }
}
```

loop

break if next = nullptr



What is the running time for iterative index? $O(n)$

No random access!

↳ Better for big data

What is the running time for recursive index? $O(n)$

↳ Bad approach for small

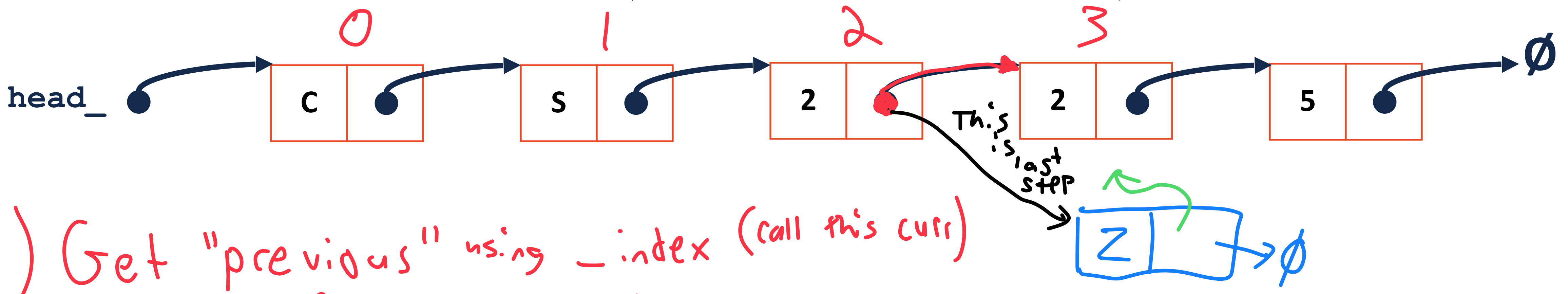
_index(n, root)

n-1, root

n-2, next

. . . .

Linked List: insert(data, index)



1) Get "previous" using _index (call this curr)
↳ As a *& so we can modify it.
↳ the red arrow is the index return

2) Create new ListNode (call this tmp)
↳ Default constructor next is null

3) Connect new node to old node @ position
↳ tmp→next = curr; ← Value of *& is pointer address!

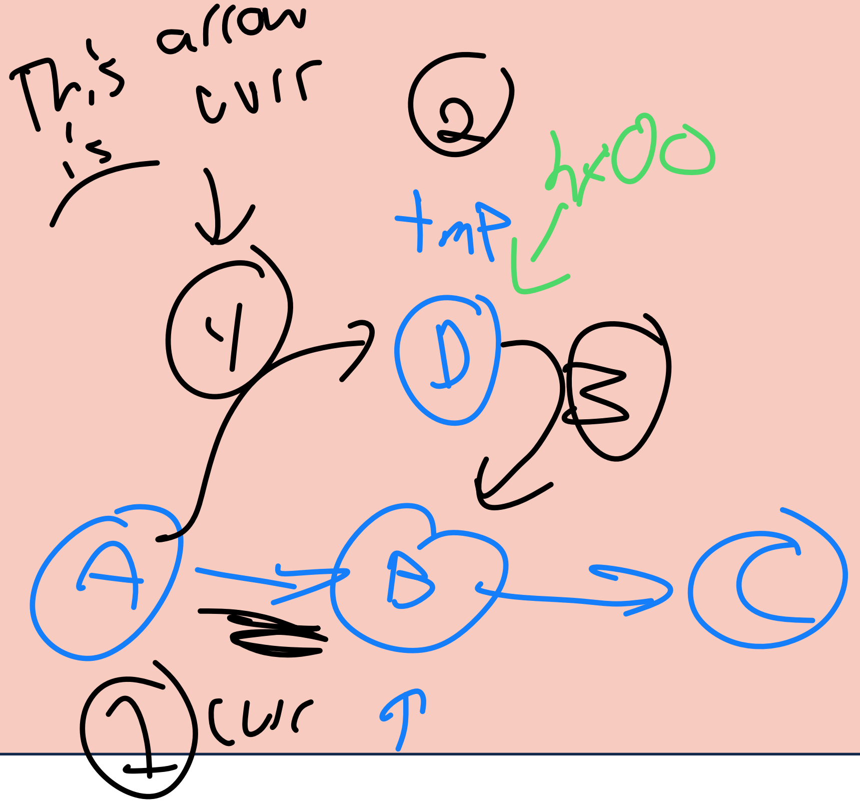
4) "previous" node's next needs to equal tmp
↳ curr = tmp;

Redo of slide for readability

List.hpp

data

```
template <typename T>
void List<T>::insertAtFront(const T& ~~t~~)
{
  ListNode *tmp = new ListNode(data);

  tmp->next = head_;

  head_ = tmp;

}
```



```
template <typename T>
void List<T>::insert(const T & data,
unsigned index) {
```

2

1) Find my index

```
1  ListNode *& curr = _index(index);
```

Make new LN data ∅

```
2  ListNode * tmp = new ListNode(data);
```

Connect new to old

```
3  tmp->next = curr;
```

Connect prev node to new node

```
4  curr = tmp;
}
```

curr is head

This is prev.next

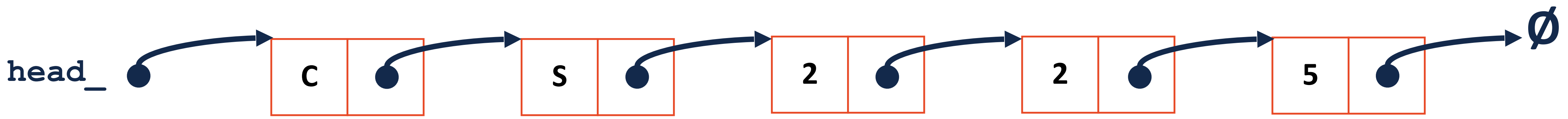
List Random Access [ ]

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

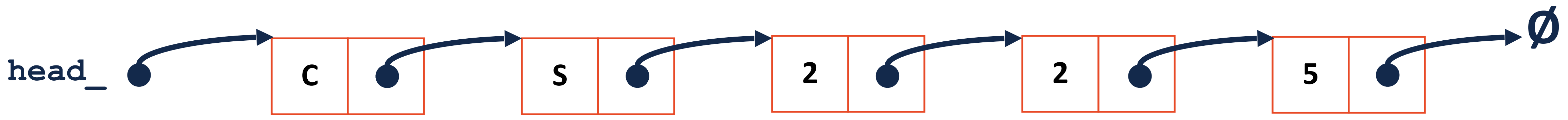
```
Did not cover yet
```

List.hpp

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









}
```

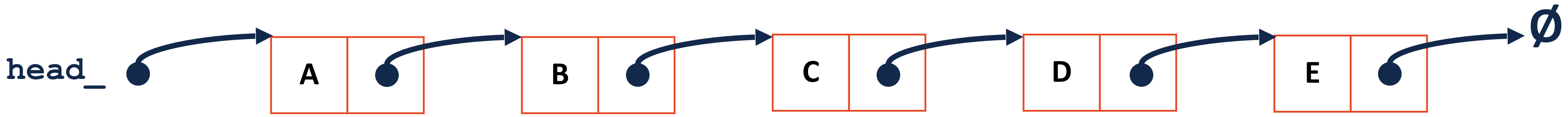


Linked List: `find(data)`

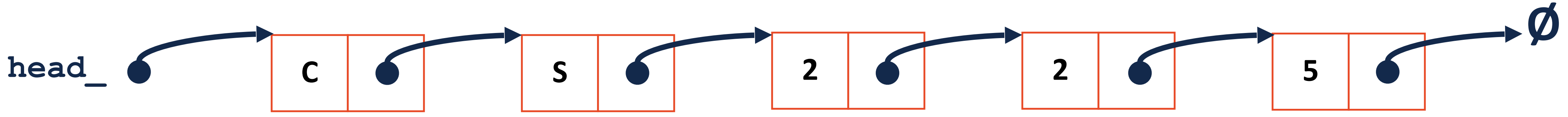


Linked List: `Remove(<parameters>)`

What input parameters make sense for remove?



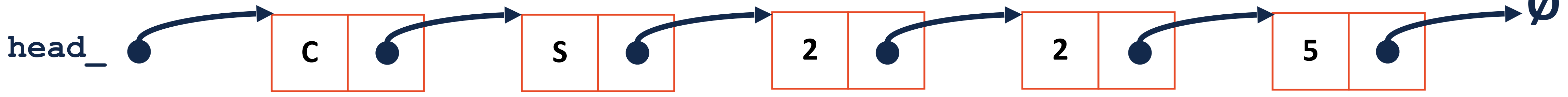
Linked List: `remove(data)`



List.hpp

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








}
```

Linked List: `remove`

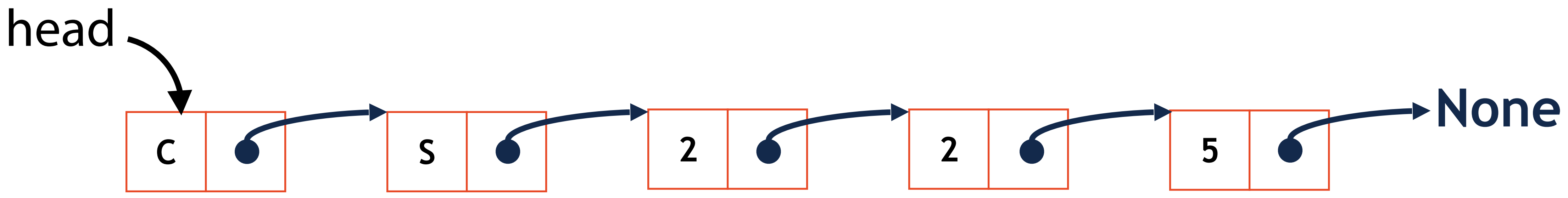


What is the running time to remove (if given a reference to a pointer)?

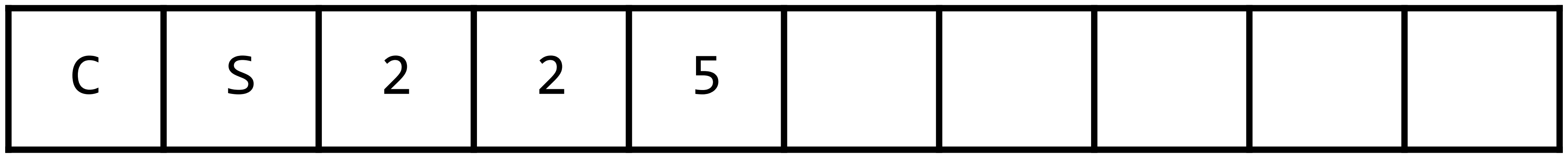
What is the running time to remove (if given a value)?

List Implementations

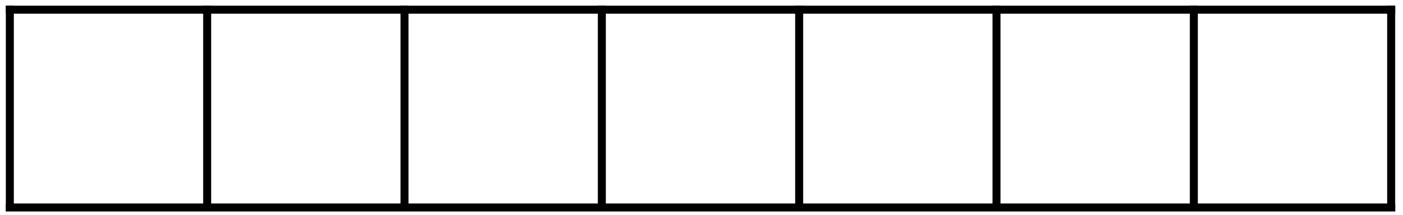
1. Linked List



2. Array List



Array List



List.h

```cpp
#pragma once

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

  T *size;

  T *capacity;

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