Data Structures

Linked Lists

CS 225

Brad Solomon

September 4, 2024

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

CS 199-225: String Algorithms and Data Structures

Logistics:

Weekly lecture followed by weekly assignment

Monday 5-5:50 PM

Siebel 0216

First lecture 9/09/24

https://courses.grainger.illinois.edu/cs225/fa2024/pages/honors.html

Syllabus has information on enrollment / HCLAs

CS 199-225: String Algorithms and Data Structures

More information:

Weekly assignment ~1-3 hours of work

'Pass' by getting 80% or above on 10 assignments

11 assignments total, drop one

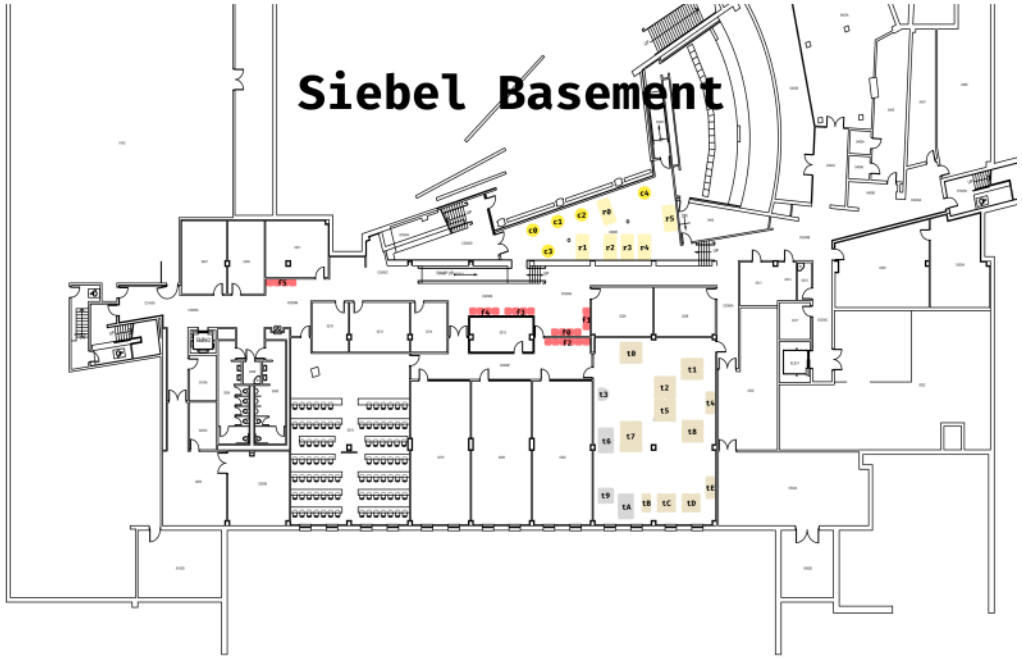
Lectures will be recorded

Two optional 'bonus lectures' with content voted by you!

Office Hour Etiquette

Schedule and link to queue on the website

Pay attention to the rules!



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

Exam 0 (9/4 — 9/6)

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

Registration started August 22

https://courses.engr.illinois.edu/cs225/fa2024/exams/

Learning Objectives

Review fundamentals of linked lists

Implement insert, index, and remove operations

Discuss pointers vs references-to-pointers

List ADT

A list is an **ordered** collection of items

Items can be either **heterogeneous** or **homogenous**

The list can be of a **fixed size** or is **resizable**

A minimal set of operations (that can be used to create all others):

1. Insert
2. Delete
3. isEmpty
4. getData
5. Create an empty list

List Implementations

1. Linked List

2. Array List

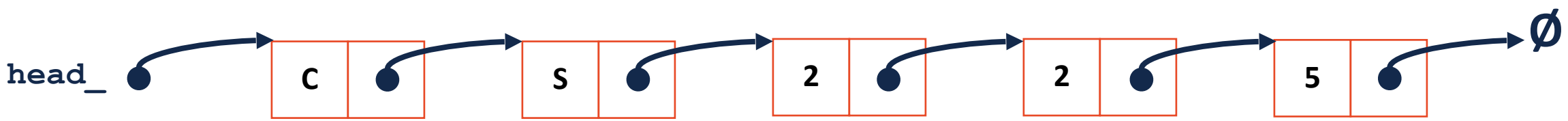
List.h

```
 1  template <class T>
 2  class List {
 3    public:
 4      /* ... */
 5    private:
…       class ListNode {
28        T & data;
29        ListNode * next;
30        ListNode(T & data) :
31         data(data), next(NULL) { }
32      };

33      ListNode *head_;
34  };
```

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

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



ListNode x ListNode y

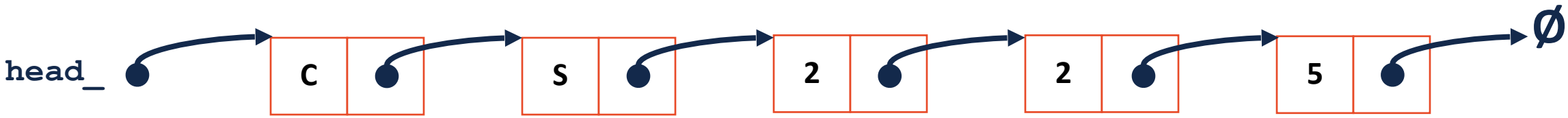
List.h

```
1  #pragma once
2
3  template <typename T>
4  class List {
5    public:
…      /* ... */
28   private:
29     class ListNode {
30       T & data;
31       ListNode * next;
32       ListNode(T & data) :
          data(data), next(NULL) { }
33     };
34
35     ListNode *head_;
36
37     /* ... */
38
39 };
…
…
79 #include "List.hpp"
```

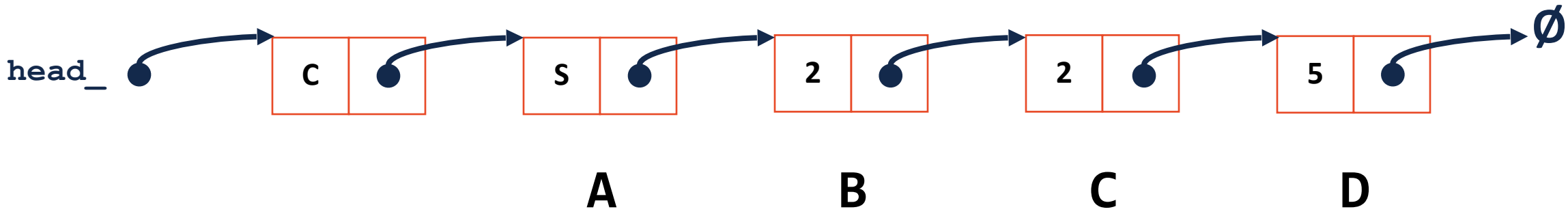
List.hpp

```
1
2
3  template <typename T>
4  void List<T>::insertAtFront(const T& t)
5  {
6
7    ListNode *tmp = new ListNode(t);
8
9
10   tmp->next = head_;
11
12
13   head_ = tmp;
14
15 }
16
17
18
19
20
21
22
```

Linked List: `insert(data, index)`



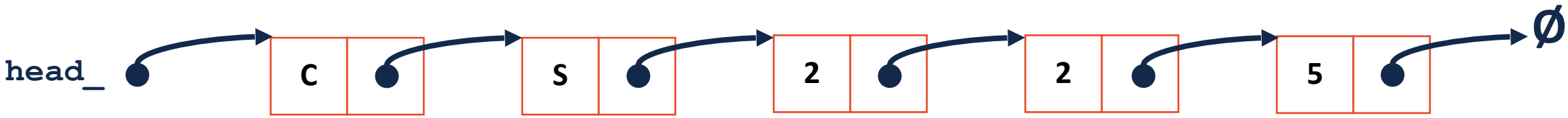
Linked List: `insert(data, index)` `insert(d,3)`



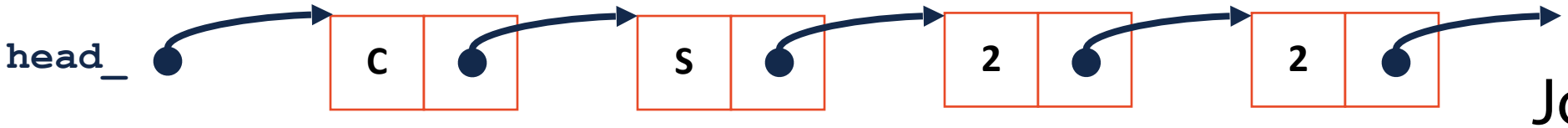
To insert a new ListNode at index 3, we need to **modify** which node?

Join Code: 225

Linked List: `_index(index)`



Linked List: `_index(index)`



Join Code: 225

What should the return type of `_index()` be?

```
[template <class T>]
```

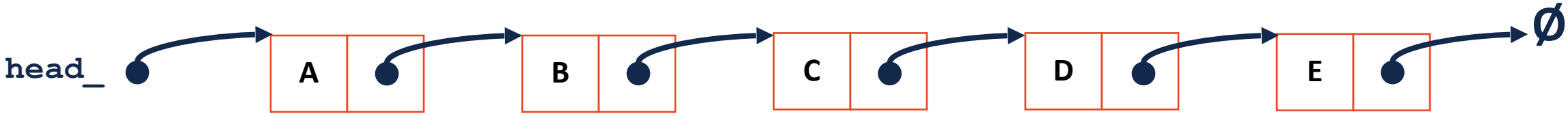
(A) `T &`

(B) `ListNode`

(C) `ListNode *`

(D) `ListNode *&`

Comparing pointer to reference-to-pointer




```
ListNode * curr = _index(3);
```

```
ListNode *& curr = _index(3);
```

A brief tangent…

List.hpp

```
58  template <typename T>
59  typename List<T>::ListNode *& List<T>::_index(unsigned index){
60          return _index(index, head_)
61  }
```

```
63  template <typename T>
64  typename List<T>::ListNode *& List<T>::_index(unsigned index, ListNode *& root){
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80  }
```

A brief tangent…

List.hpp

```
58  template <typename T>
59  typename List<T>::ListNode *& List<T>::_index(unsigned index){
60          return _index(index, head_)
61  }
```

```
63  template <typename T>
64  typename List<T>::ListNode *& List<T>::_index(unsigned index, ListNode *& root){
65
66
67
68      if (index == 0){ return root; }
69
70
71
72      if (root == nullptr){ return root; }
73
74
75
76      return _index(index - 1, root -> next);
77
78
79
80  }
```

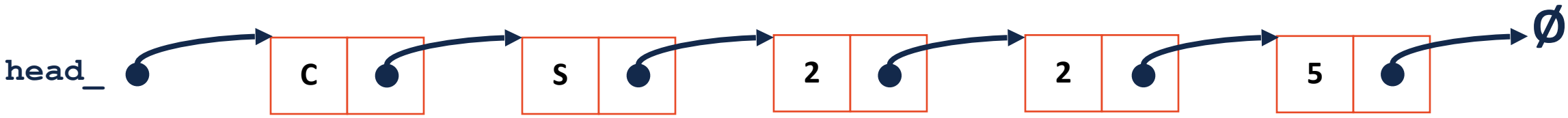
A brief tangent...

List.hpp

```
1  // Iterative Solution:
2  template <typename T>
3  typename List<T>::ListNode *& List<T>::_index(unsigned index) {
4    if (index == 0) { return head; }
5    else {
6      ListNode *curr = head;
7      for (unsigned i = 0; i < index - 1; i++) {
8         curr = curr->next;
9      }
10     return curr->next;
11   }
12 }
```

Which solution is better (iterative or recursive)?

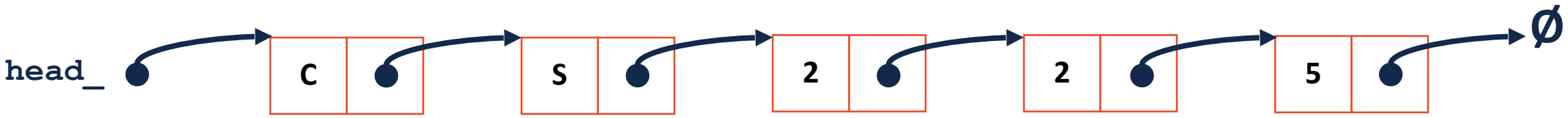
Linked List: `insert(data, index)`



1) Get reference to previous node's next

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

Linked List: `insert(data, index)`



1) Get reference to previous node's next

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

2) Create new ListNode

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

3) Update new ListNode's next

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

4) Modify the previous node to point to new ListNode

```
curr = tmp;
```

Lets compare…

```
1
2  template <typename T>
3  void List<T>::insertAtFront(const T& t)
4  {
5    ListNode *tmp = new ListNode(t);
6
7    tmp->next = head_;
8
9    head_ = tmp;
10
11 }
12
13
14
15
16
17
18
19
20
21
22
```

List.hpp

```
1
2  template <typename T>
3  void List<T>::insert(const T & data,
4  unsigned index) {
5
6
7
8    ListNode *& curr = _index(index);
9
10
11
12   ListNode * tmp = new ListNode(data);
13
14
15
16   tmp->next = curr;
17
18
19
20   curr = tmp;
21 }
22
```

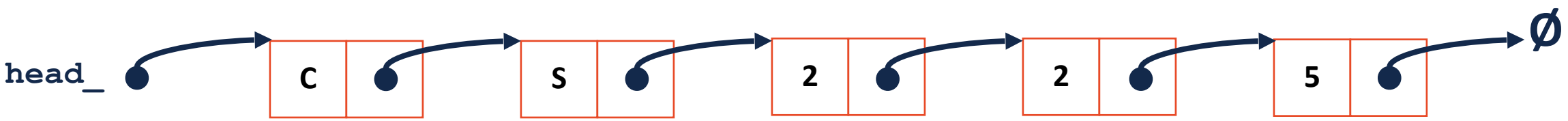
List Random Access [ ]

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

What return type should this function have?

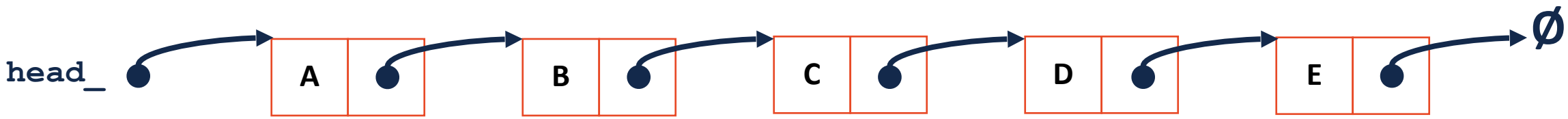
List.hpp

```
48 template <typename T>
49 T & List<T>::operator[](unsigned index) {
50
51
52
53
54
55
56
57
58 }
```

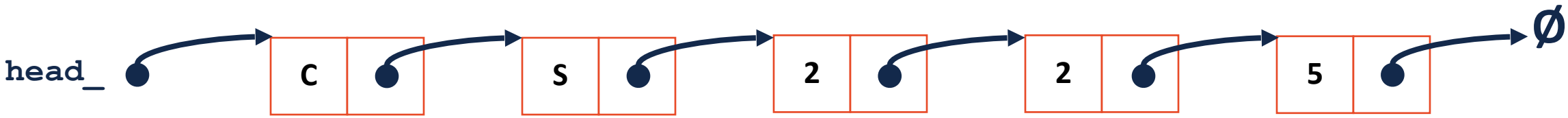


Linked List: `remove(<parameters>)`

What input parameters make sense for remove?

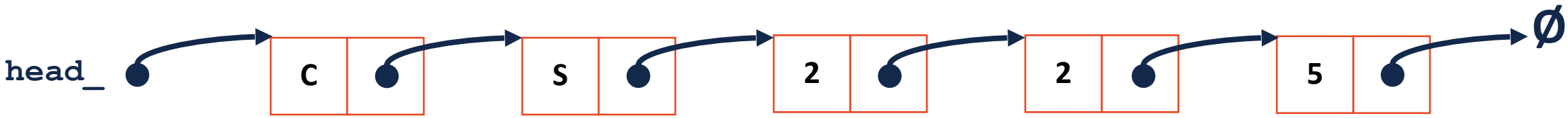


Linked List: `remove(ListNode *& n)`

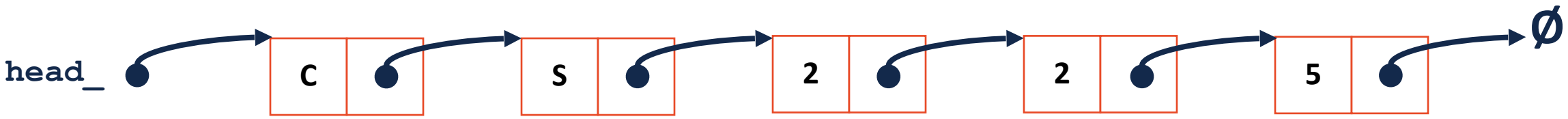


```
103 template <typename T>
104 T List<T>::remove(ListNode *& node) {
105
106
107
108
109
110
111
112 }
```

Linked List: `remove(T & data)`



Linked List: `remove`

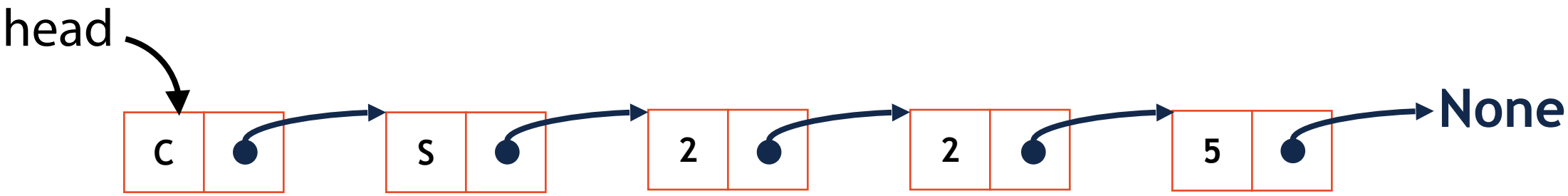


Running time for **`remove(ListNode *&)`**

Running time for **`remove(T & data)`**

List Implementations

1. Linked List



2. Array List