

CS 225

Data Structures

Feb. 7 – List Implementation
Wade Fagen-Ulmschneider

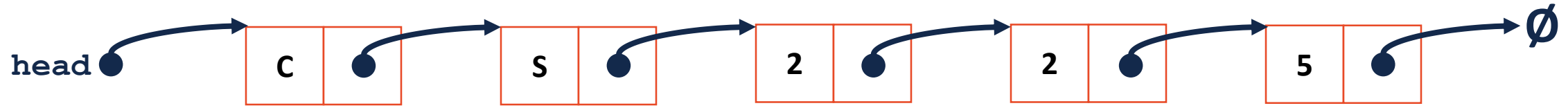
List.h

```
1 #ifndef LIST_H_
2 #define LIST_H_
3
4 template <typename T>
5 class List {
6     public:
7         /* ... */
8
9     private:
10        class ListNode {
11            public:
12                T & data;
13                ListNode * next;
14                ListNode(T & data) :
15                    data(data), next(NULL) { }
16
17        };
18
19        ListNode *head_;
20
21 };
22
23 #endif
```

List.cpp

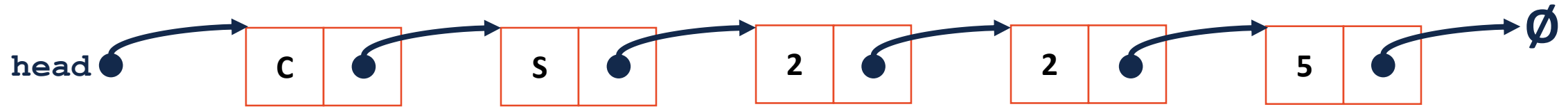
```
1 #include "List.h"
2
3 template <typename T>
4 void List::insertAtFront(const T& t) {
5     ListNode *e = new ListNode(t);
6     e->next = head_;
7     head_ = e;
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22 }
```

Linked Memory



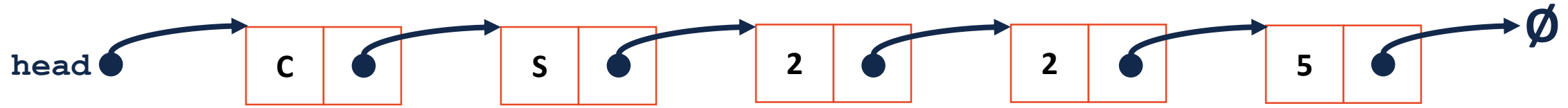
```
1 #include "List.h"
2
3 ListNode *& List::_find(unsigned index) const {
4
5
6
7
8
9
10 }
```

Linked Memory



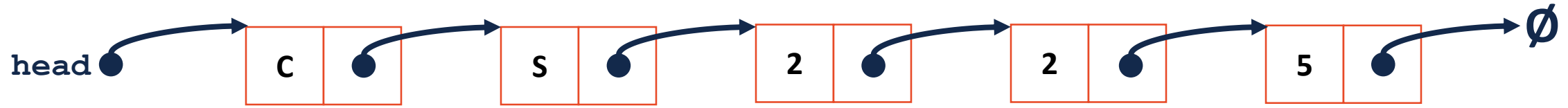
```
1 #include "List.h"
2
3 ListNode *& List::_find(unsigned index) const {
4     if (index == 0) { return head; }
5     else {
6         ListNode *thru = head;
7         for (unsigned i = 0; i < index - 1; i++) {
8             thru = thru->next;
9         }
10        return thru->next;
11    }
12 }
```

Linked Memory



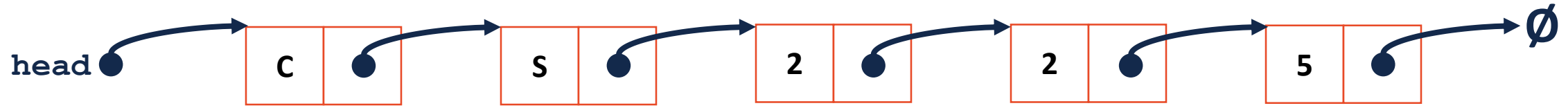
```
1 #include "List.h"
2
3 ListNode *& List::_find(unsigned index) const {
4     if (index == 0) { return head; }
5     else {
6         ListNode *thru = head;
7         for (unsigned i = 0; i < index - 1; i++) {
8             thru = thru->next;
9         }
10        return thru->next;
11    }
12 }
13
14 template <typename T>
15 T & List::get(unsigned index) const {
16
17
18
19
20
21
22 }
```


Linked Memory



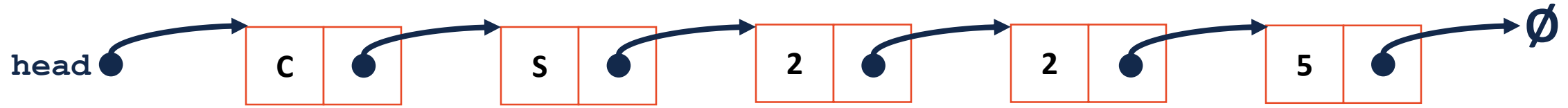
```
1 #include "List.h"
2
3 ListNode *& List::_find(unsigned index) const {
4     if (index == 0) { return head; }
5     else {
6         ListNode *thru = head;
7         for (unsigned i = 0; i < index - 1; i++) {
8             thru = thru->next;
9         }
10        return thru->next;
11    }
12 }
13
14 template <typename T>
15 T & List::insert(T & t, unsigned index) {
16
17
18
19
20
21
22 }
```

Linked Memory

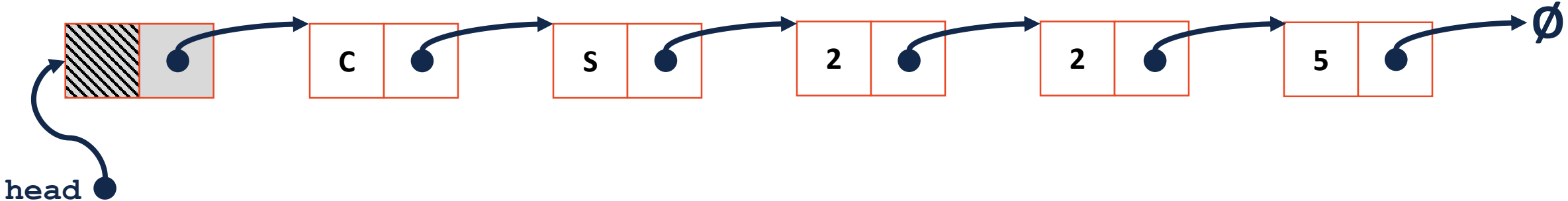


```
1 #include "List.h"
2
3 ListNode *& List::_find(unsigned index) const {
4     if (index == 0) { return head; }
5     else {
6         ListNode *thru = head;
7         for (unsigned i = 0; i < index - 1; i++) {
8             thru = thru->next;
9         }
10        return thru->next;
11    }
12 }
13
14 template <typename T>
15 T & List::remove(unsigned index) {
16
17
18
19
20
21
22 }
```

Linked Memory



Sentinel Node





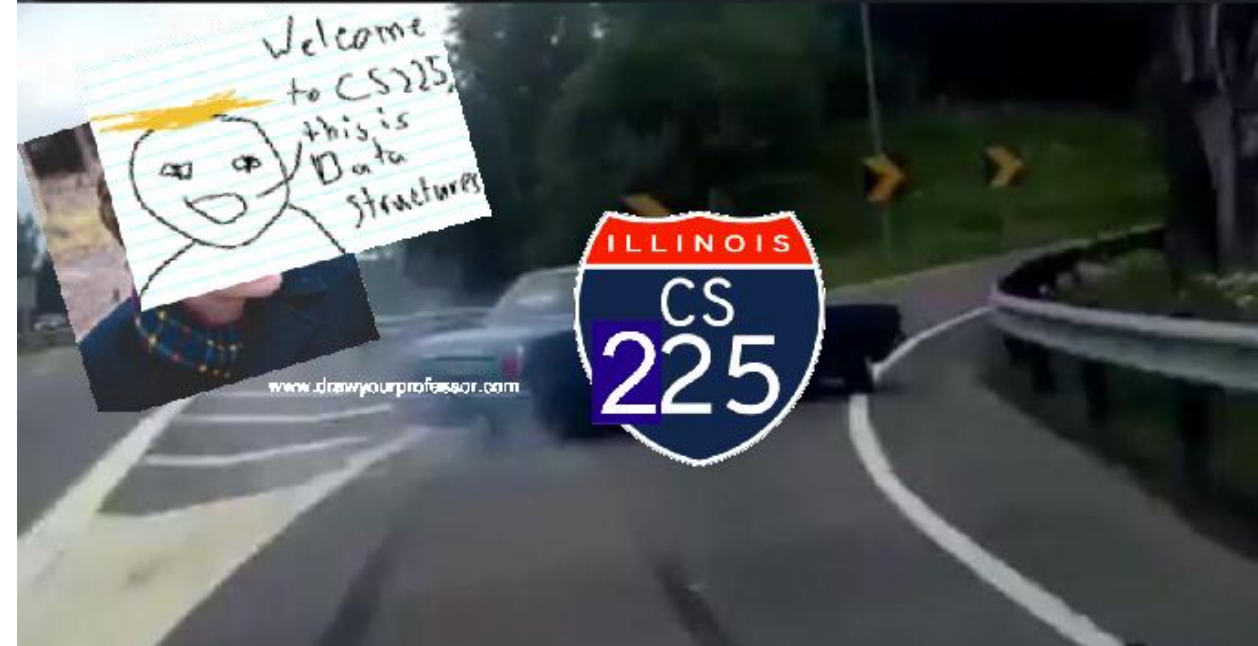
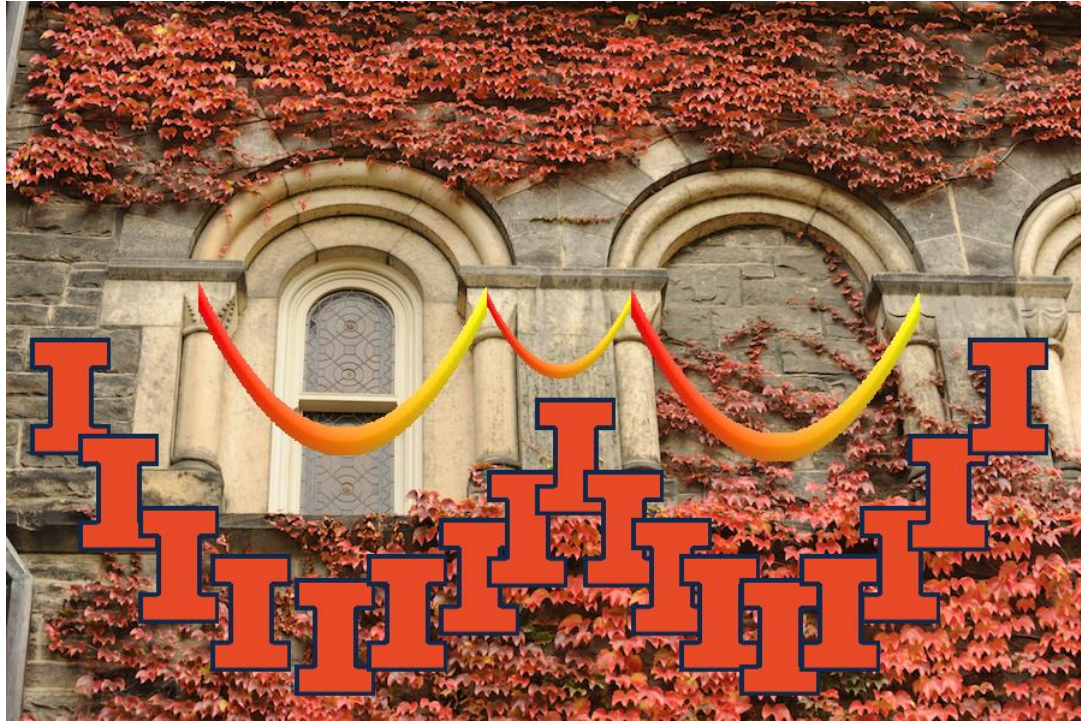
MP2

MP2

The image is a collage with several elements:

- Smartphone:** An iPhone-style smartphone is shown in the upper left, tilted, displaying a home screen with various app icons.
- Lecture Hall:** The background is a photograph of a lecture hall filled with students sitting at desks, viewed from behind.
- Text 'ring ring!':** Large, black, outlined text with a green exclamation point is positioned in the upper right.
- Text 'WHO IS IT?':** Large, white, bold text is centered in the middle of the image.
- Text 'it's a...':** Large, yellow, bold text is positioned in the lower left.
- Diagram:** A white box in the lower right contains a diagram with two boxes: 'address' (labeled 'pointer' below) and 'value' (labeled 'variable' below). An arrow points from 'address' to 'value'.

MP2



MP2



MP2



List Implementations

1. Linked List

2.

```
1 #ifndef LIST_H
2 #define LIST_H
3
4 template <typename T>
5 class List {
6     public:
7         /* ... */
8     private:
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40 };
41
42 #endif
```

Array Implementation

c	s	2	2	5
[0]	[1]	[2]	[3]	[4]

Array Implementation

insertAtFront:

C	S	2	2	5
[0]	[1]	[2]	[3]	[4]

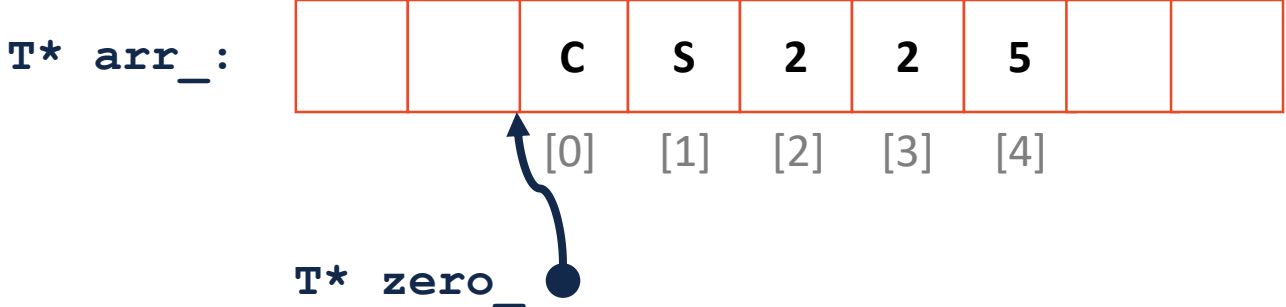
Resize Strategy – Details



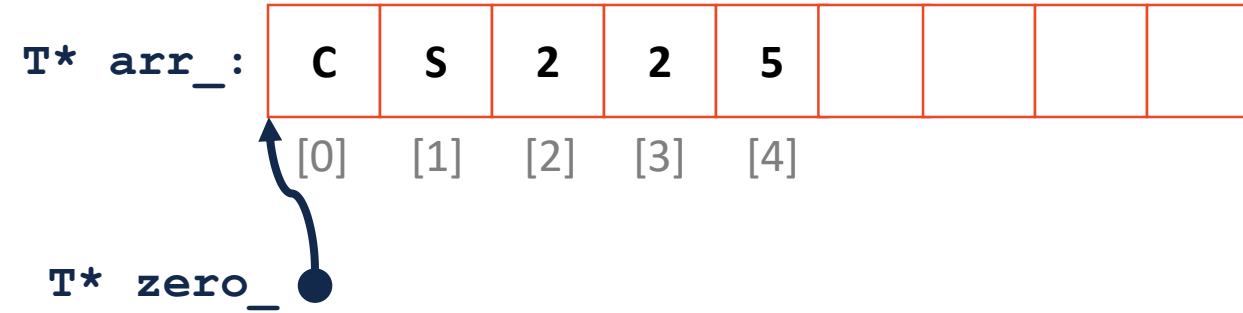
Resize Strategy – Details



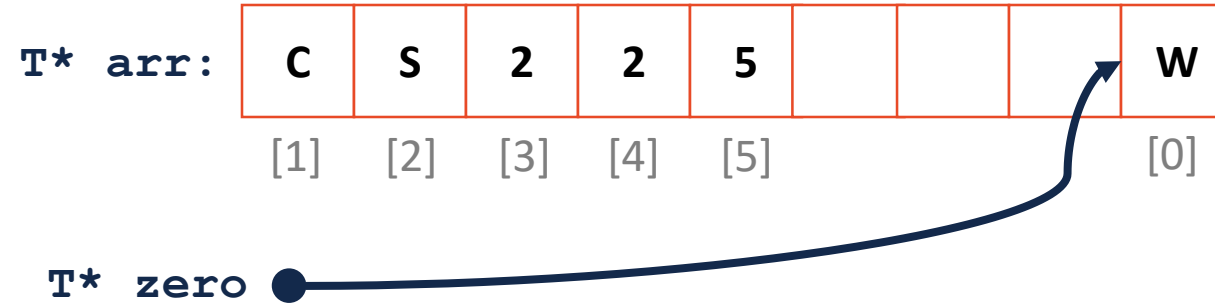
Array Implementation



Array Implementation

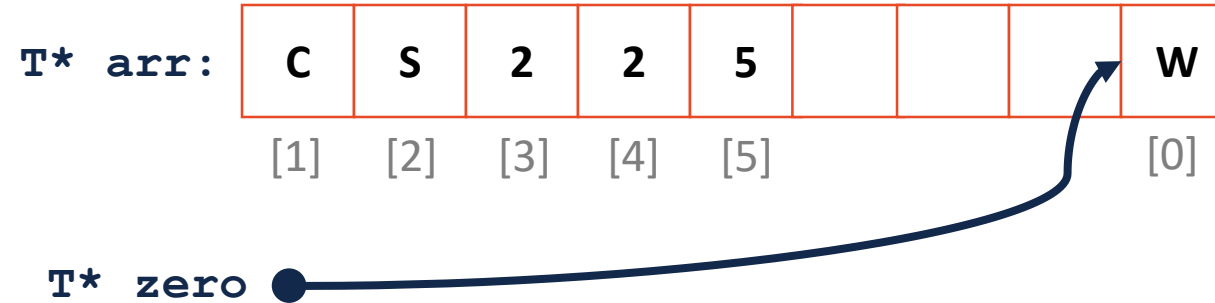


Array Implementation



```
21 ListNode *& List::_get(unsigned index) const {  
22     return arr_ [ (zero_ - arr_) + index % capacity_ ] ;  
23 }
```

Array Implementation



```
21 ListNode *& List::_get(unsigned index) const {  
22     return arr_ [ (zero_ - arr_) + index % capacity_ ] ;  
23 }
```

Array Implementation

	Singly Linked List	Array
Insert/Remove at front		
Insert at given element		
Remove at given element		
Insert at arbitrary location		
Remove at arbitrary location		