CS 225

Data Structures

September 15 – Templates and Lists

G Carl Evans

Templates

template1.cpp

```
1
2
3  T maximum(T a, T b) {
4    T result;
5    result = (a > b) ? a : b;
6    return result;
7  }
```

List.h

```
1  #pragma once
2
3
4  class List {
5    public:
6
7
8
9
10
11
12
13
14   private:
15
16
17
18 };
19
20
21
22
```

List.hpp

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
```

List ADT to C++ Interface

- **Insert – `void insert(const T &data);`**
- **Delete – `void delete();`**
- **Get Data – `T getData() const;`**
- **Is Empty – `bool isEmpty() const;`**
- **Create Empty List – `List();`**

List Implementations

1.

2.

Linked Memory

List.h

```
28 class ListNode {
29   T  data;
30   ListNode * next;
31   ListNode(T & data) : data(data), next(NULL) { }
32 };
```

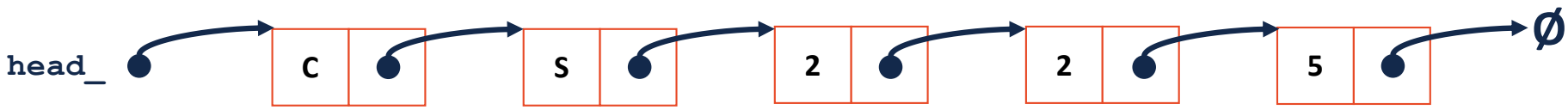
List.h

```
 1  #pragma once
 2
 3  template <typename T>
 4  class List {
 5    public:
…      /* ... */
19
20    private:
21      class ListNode {
22        public:
23          T data;
24          ListNode * next;
25          ListNode(const T & data) :
            data(data), next(NULL) { }
26      };
27
28      ListNode *head_;
…
      };
```

List.hpp

```
 9  #include "List.h"
…
14  template <typename T>
15  void List::insertAtFront(const T& d) {
16
17
18
19
20
21
22  }
```

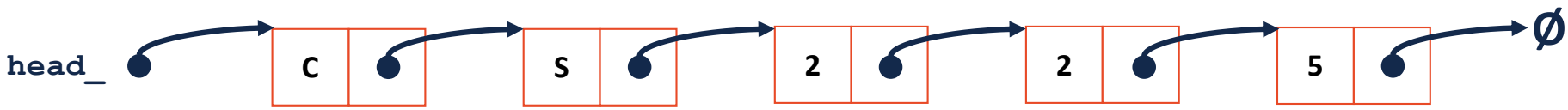
Linked Memory



List.hpp

```
57  template <typename T>
58  typename List<T>::ListNode *&
      List<T>::_index(unsigned index) {
59
60
61  }
62
63
64
65
```

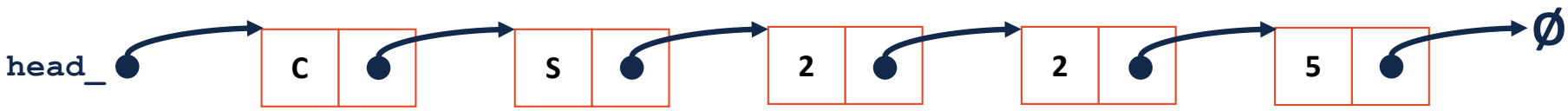
Linked Memory



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;
  }
}
```

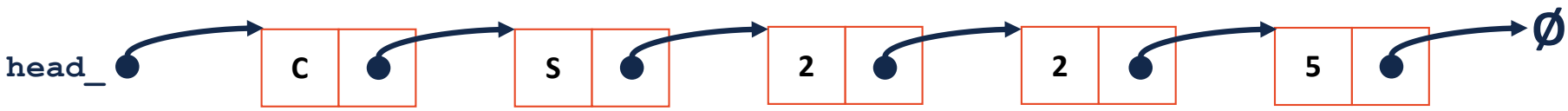
Linked Memory



List.cpp

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

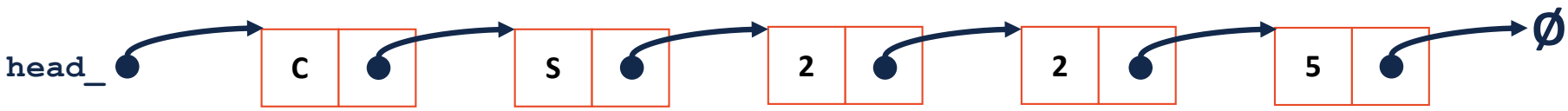
Linked Memory



List.cpp

```
90  template <typename T>
91  void List<T>::insert(const T & t, unsigned index) {
92
93
94
95
96
97
98
99  }
```

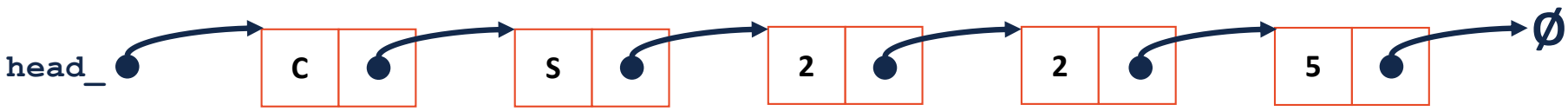
Linked Memory



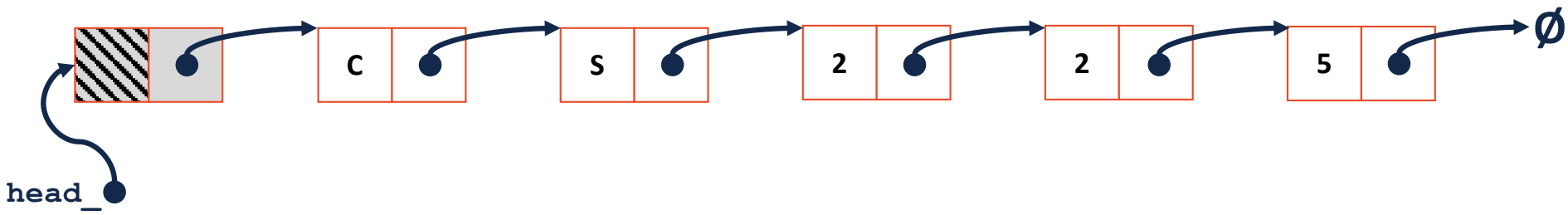
List.cpp

```
103 template <typename T>
104 T List<T>::remove(unsigned index) {
105
106
107
108
109
110
111
112 }
```

Linked Memory



Sentinel Node



List Implementations

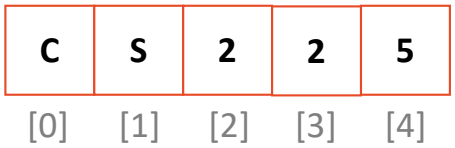
1. **Linked List**

2.

List.h

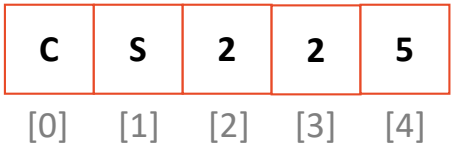
```
 1 #pragma once
 2
 3 template <typename T>
 4 class List {
 5   public:
…      /* ... */
28   private:
29
30
31
32
33
34
35
36
37
38
39
40
41
42 };
```

Array Implementation

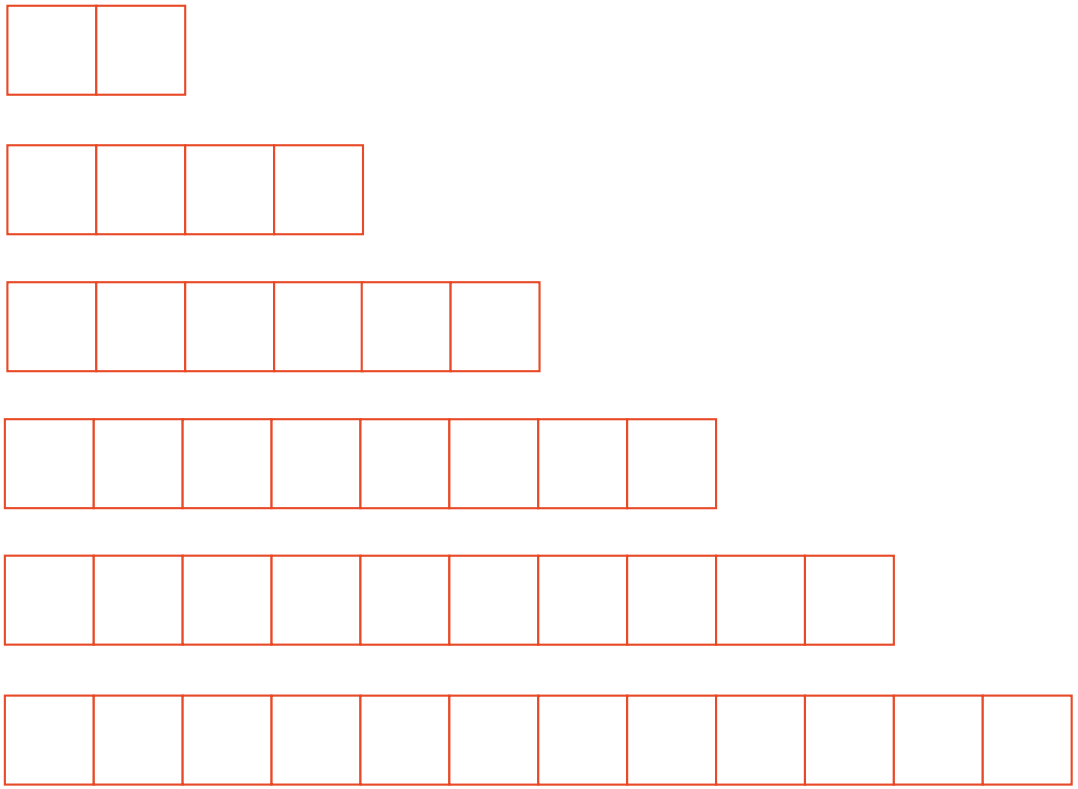


Array Implementation

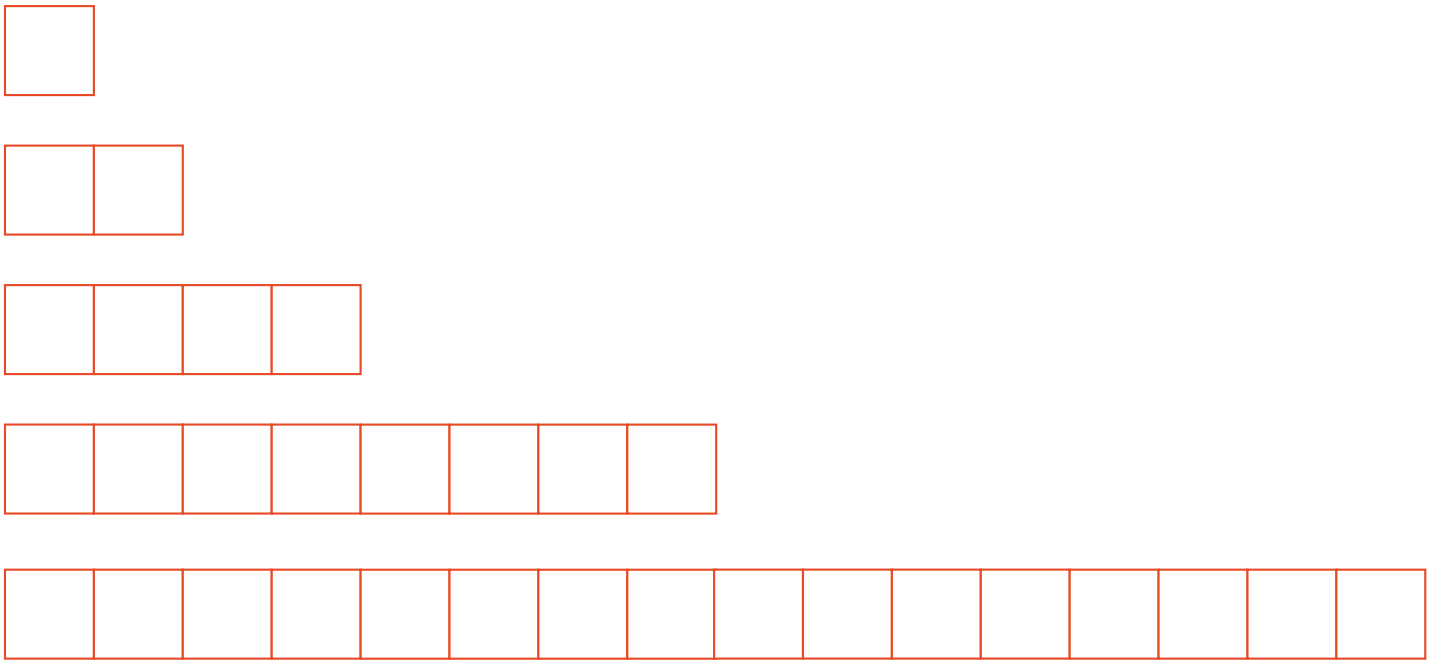
insertAtFront:



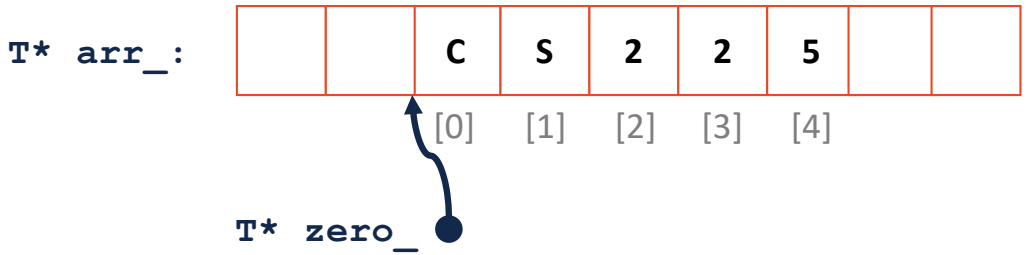
Resize Strategy – Details



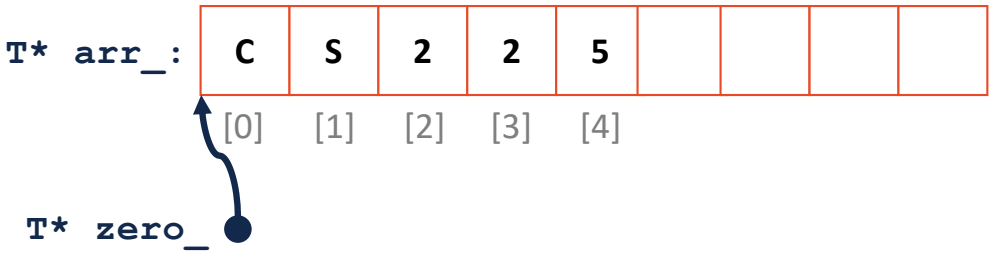
Resize Strategy – Details



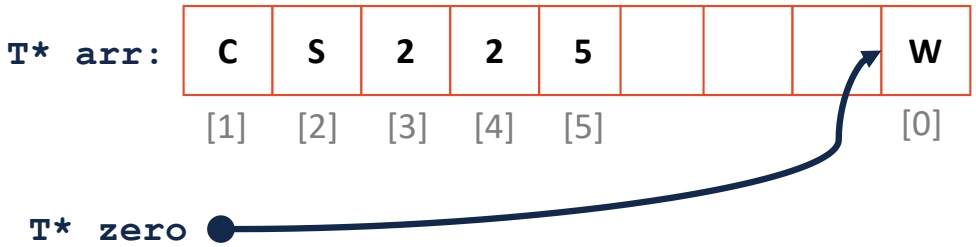
Array Implementation



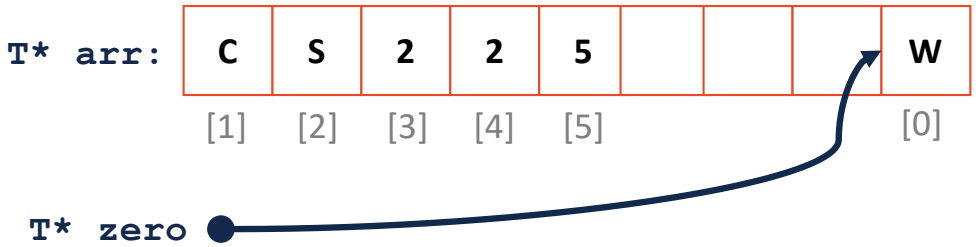
Array Implementation



Array Implementation



Array Implementation



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		