CS 225

Data Structures

February 7 – Lists 2

G Carl Evans

mp_stickers

- Due today at 11:59pm

- Must be in your master branch on
  https://github-dev.cs.illinois.edu/cs225-sp22/

- 24 hour extension twice in the semester for the MPs

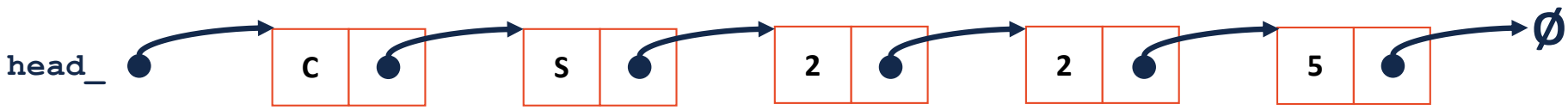
List.h

```
 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

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

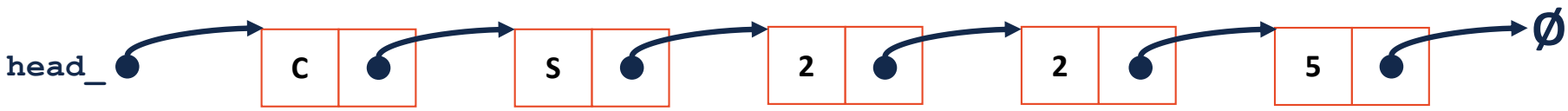
Linked Memory



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;
  }
}
```

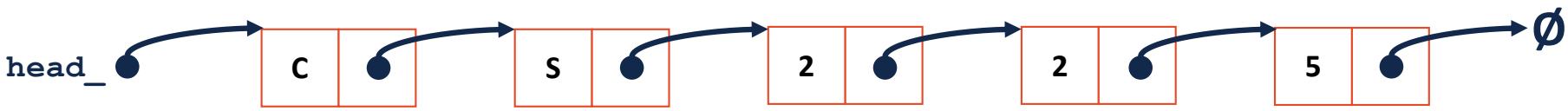
Linked Memory



List.cpp

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

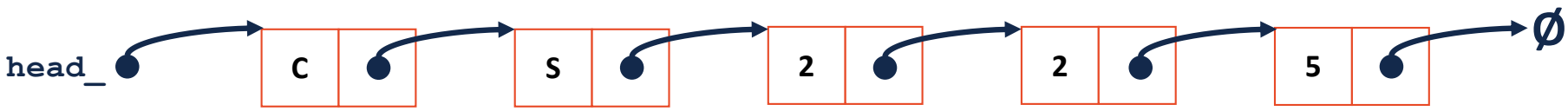
Linked Memory



List.cpp

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

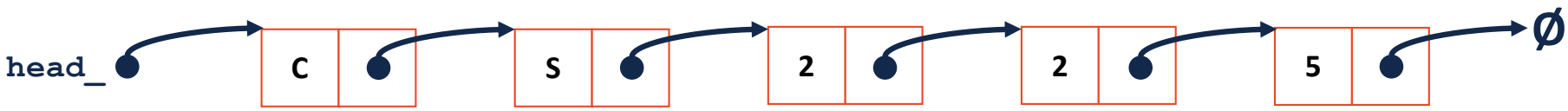
Linked Memory



List.cpp

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

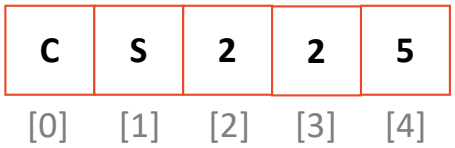
Linked Memory



List.h

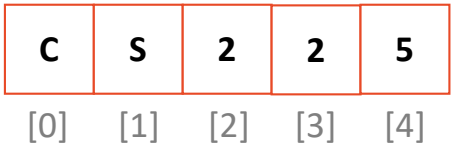
```
 1 #pragma once
 2
 3 template <typename T>
 4 class List {
 5   public:
…      /* ... */
28   private:
29
30
31
32
33 };
```

Array Implementation

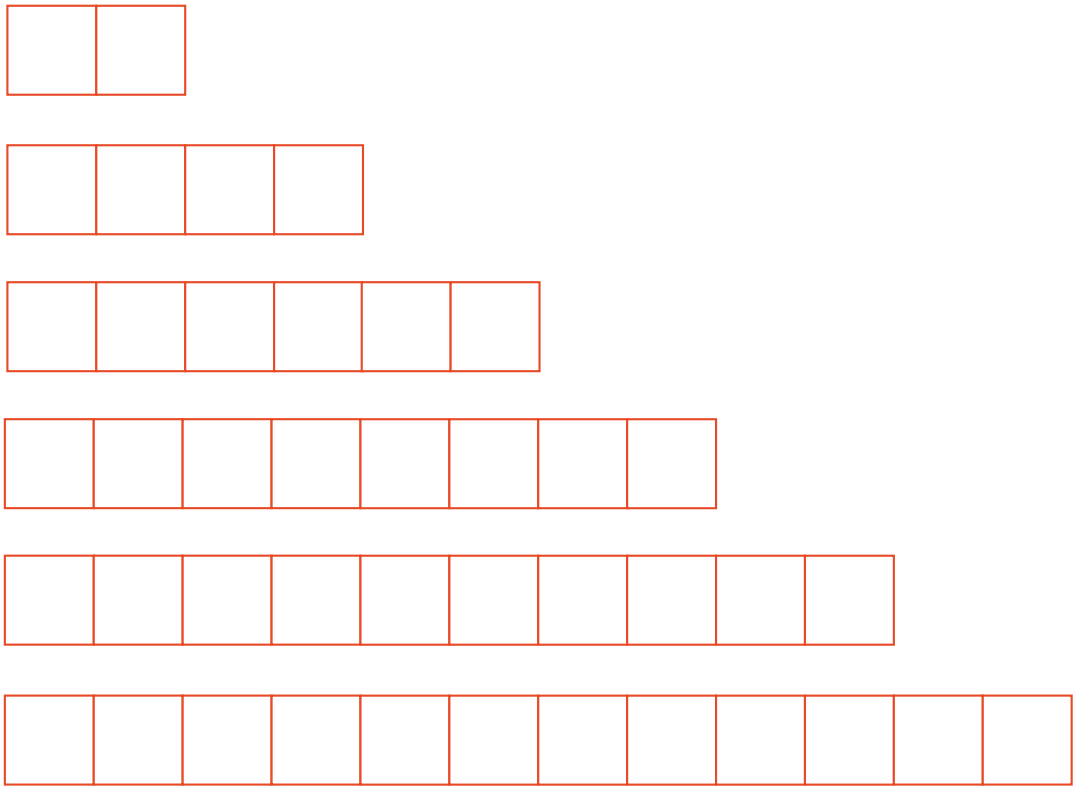


Array Implementation

insertAtFront:

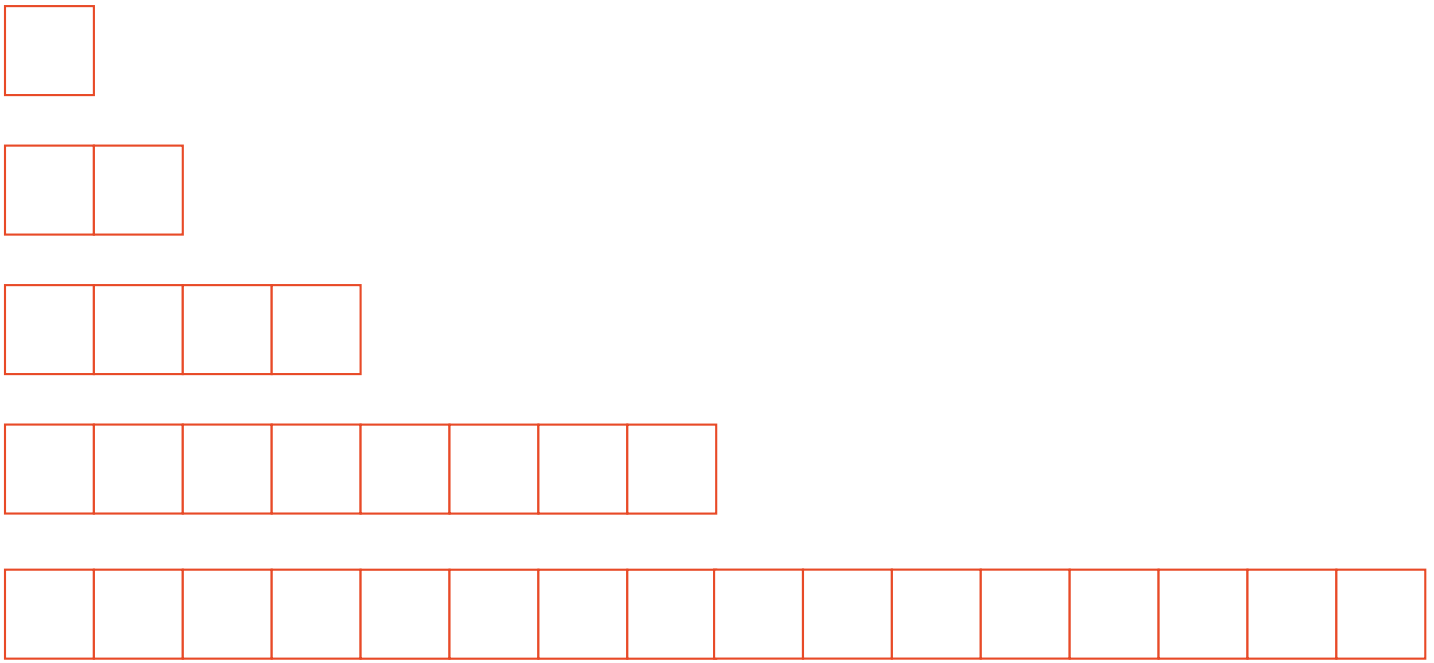


Resize Strategy: +2 elements every time



Resize Strategy: +2 elements every time

Resize Strategy: x2 elements every time



Resize Strategy: x2 elements every time

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		

std::vector

Stack ADT

Queue ADT

Stack.h

```
 1 #pragma once
 2
 3 #include <vector>
 4
 5 template <typename T>
 6 class Stack {
 7   public:
 8     void push(const T & t);
 9     T pop();
10     bool isEmpty();
11
12   private:
13     std::vector<T> list_;
14 };
15
16 #include "Stack.hpp"
```

Stack Implementation

```
 3 template <typename T>
 4 void Stack<T>::push(const T & d) {
 5   list_.push_back(d);
 6 }
 7
 8 template <typename T>
 9 T Stack<T>::pop() {
10   T data = list_.back();
11   list_.pop_back();
12   return data;
13 }
```

Implications of Design

1.

```
class ListNode {
  public:
    T & data;
    ListNode * next;
    …
```

2.

```
class ListNode {
  public:
    T * data;   …
```

3.

```
class ListNode {
  public:
    T data;   …
```

Implications of Design

	Storage by Reference	Storage by Pointer	Storage by Value
Who manages the lifecycle of the data?			
Is it possible for the data structure to store NULL?			
If the data is manipulated by user code while in our data structure, is the change reflected in our data structure?			
Is it possible to store literals?			
Speed			

Data Lifecycle

Storage by reference:

```
1  Sphere s;
2  myStack.push(s);
```

Storage by pointer:

```
1  Sphere s;
2  myStack.push(&s);
```

Storage by value:

```
1  Sphere s;
2  myStack.push(s);
```

Possible to store NULL?

Storage by reference:

```
class ListNode {
  public:
    T & data;
    ListNode * next;
    ListNode(T & data) : data(data), next(NULL) { }
};
```

Storage by pointer:

```
T ** arr;
```

Storage by value:

```
T * arr;
```

Data Modifications

```
1  Sphere s(1);
2  myStack.push(s);
3
4  s.setRadius(42);
5
6  Sphere r = myStack.pop();
7  // What is r's radius?
```

Speed