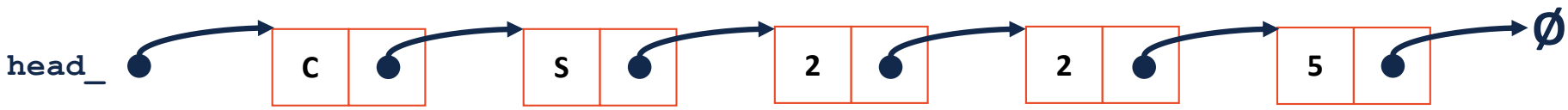
CS 225

Data Structures

January 24 – Linked List Implementation

G Carl Evans

Linked Memory



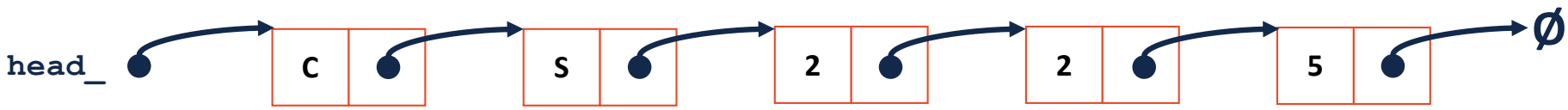
List.h

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

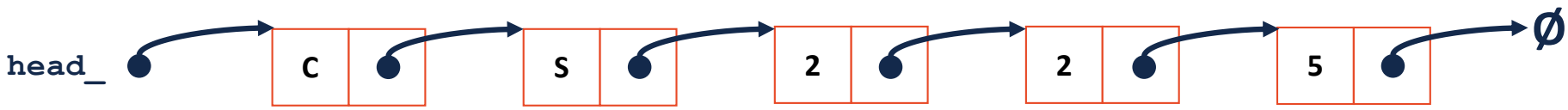
List.hpp

```
 1
 2  template <class T>
 3  void List<T>::insertAtFront(const T& t) {
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
16
17
18
19
20
21  }
22
```

Linked Memory : `insert(data,index)`



Linked Memory : `_index`



List.hpp

```
33 ListNode *& List<T>::_index(unsigned index) {
34         return _index(index, head_)
35 }
```

```
33 ListNode *& List<T>::_index(unsigned index,ListNode *&head) {
34         if(head == nullptr) {
35                 return head;
36         } else {
37         return _index( index - 1, head->next);
38         }
39 }
40
```

List.hpp

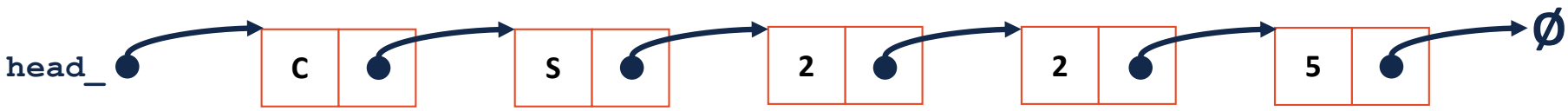
```
// Iterative Solution:
template <typename T>
typename List<T>::ListNode *& List<T>::_index(unsigned index) {
  if (index == 0) { return head_; }
  else {
    ListNode *thru = head_;
    for (unsigned i = 0; i < index - 1; i++) {
       thru = thru->next;
    }
    return thru->next;
  }
}
```

Running Time of Linked List `_index`

Recursive

Iterative

Linked Memory: `insert`

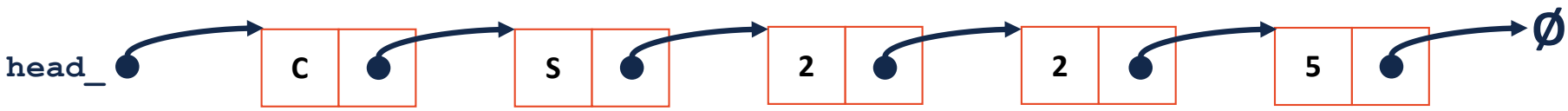


List.hpp

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

Running Time of Linked List `insert`

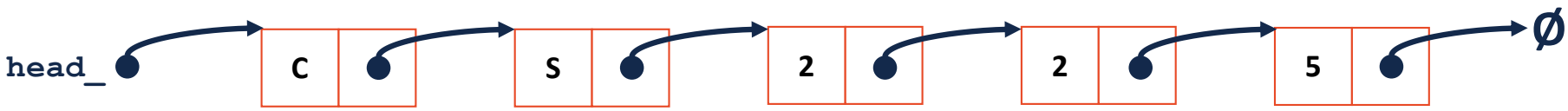
Linked Memory: `operator[]`



List.hpp

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

Linked Memory: `remove`



List.hpp

```
109  template <typename T>
110  T List<T>::remove(unsigned index) {
  …
     }
```

Linked Memory Runtimes