CS 225

Data Structures

January 27 – Linked List Implementation

G Carl Evans

Linked Lists

List.h

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

List.hpp

```
58 template <typename T>
59 typename List<T>::ListNode *& List<T>::_index(unsigned index) {
60   return _index(index, head_);
62 }
```

```
63 template <typename T>
64 typename List<T>::ListNode *& List<T>::_index(unsigned index, ListNode *& node)
65 {
66   if (index == 0 || node == nullptr)
67   {
68     return node;
69   } else {
70     return _index(index - 1, node->next);
71   }
72 }
```

List.hpp

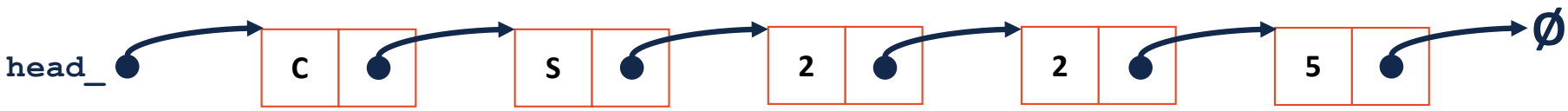
```
// Iterative Solution:
template <typename T>
typename List<T>::ListNode *& List<T>::_index(unsigned index) {
  if (index == 0) { return head; }
  else {
    ListNode *thru = head;
    for (unsigned i = 0; i < index - 1; i++) {
       thru = thru->next;
    }
    return thru->next;
  }
}
```

Running Time of Linked List `_index`

Recursive

Iterative

Linked Memory: `insert`

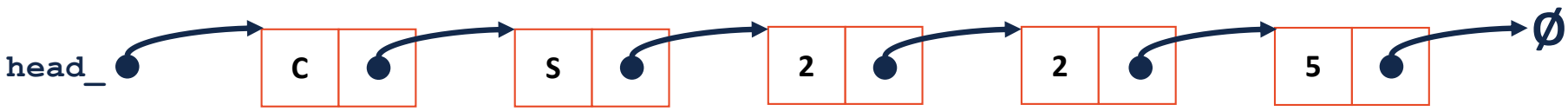


List.hpp

```
96  template <typename T>
97  void List<T>::insert(const T & data, unsigned index) {
 …
    }
```

Running Time of Linked List `insert`

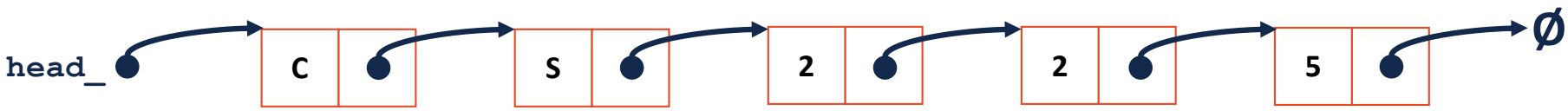
Linked Memory: `operator[]`



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