

Data Structures

Lists and List ADT

CS 225

August 25, 2023

Brad Solomon & G Carl Evans



UNIVERSITY OF
ILLINOIS
URBANA - CHAMPAIGN

Department of Computer Science

Learning Objectives

Define the functions and operations of the List ADT

Discuss list implementation strategies

Explore how to code and use a linked list

Practice fundamentals of C++ in the context of lists

Pointer-to-constant vs constant pointer

```
1 int x = 3;
2 int y = 2;
3 // *** A ***
4 const int* a = &x;
5
6
7 a = &y;
8
9 // *** B ***
10 const int* b = &x;
11
12 *b = y;
13
14
15 // *** C ***
16 int* const c = &x;
17
18 c = &y;
19
20
21 // *** D ***
22 int* const d = &x;
23
24 *d = y;
25
```

(const int)* a = &x;

(const int)* b = &x;

(int*) const c = &x;

(int*) const d = &x;

What types of “stuff” do we want in our list?

--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--

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 }
```

Abstract Data Types

A way of describing a data type as a combination of:

Data being stored by the data type

Operations that can be performed on the data type

The actual implementation details of the ADT aren't relevant!

List ADT (What do we want our list to do?)

List ADT



A list is an **ordered** collection of items

Items can be either **heterogeneous** or **homogenous**

The list can be of a **fixed size** or is **resizable**

A minimal set of operations (that can be used to create all others):

1. Insert
2. Delete
3. isEmpty
4. getData
5. Create an empty list

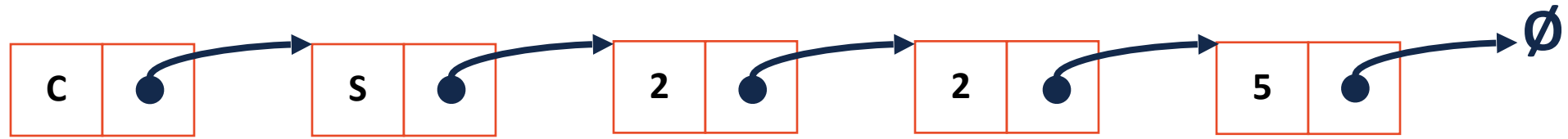


List Implementations

1.

2.

Linked List



List.h

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

List.h

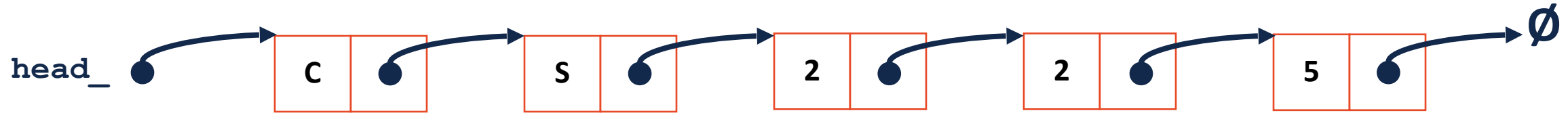
```
1 #pragma once
2
3 template <class T>
4 class List {
5     public:
6         /* ... */
7     ...
8     private:
9         class ListNode {
10             T & data;
11             ListNode * next;
12             ListNode(T & data) :
13                 data(data), next(NULL) { }
14         };
15
16         ListNode *head_;
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     ...
43
44     ...
45
46     ...
47
48     ...
49
50     ...
51
52     ...
53
54     ...
55
56     ...
57
58     ...
59
60     ...
61
62     ...
63
64     ...
65
66     ...
67
68     ...
69 #include "List.hpp" // **A**
```

List.hpp



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

Linked List: insertAtFront(data)



List.h

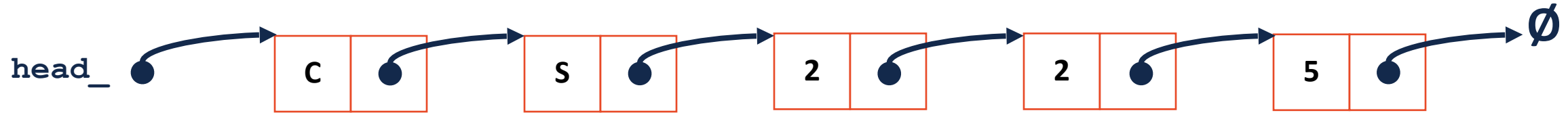
```
1 #pragma once
2
3 template <class T>
4 class List {
5     public:
6         /* ... */
7     ...
8     private:
9         class ListNode {
10             T & data;
11             ListNode * next;
12             ListNode(T & data) :
13                 data(data), next(NULL) { }
14         };
15
16         ListNode *head_;
17
18         /* ... */
19     };
20
21     ...
22     ...
23
24 79 #include "List.hpp"
```

List.hpp



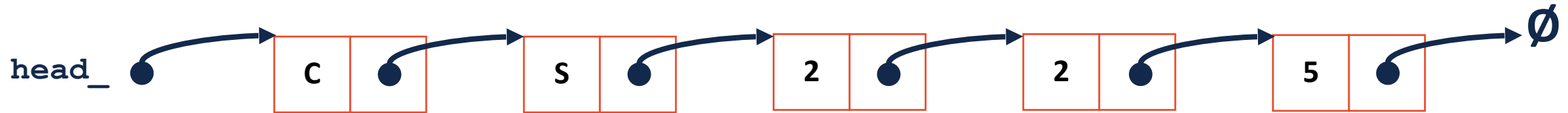
```
1
2
3 template <typename T>
4 void List<T>::insertAtFront(const T& t)
5 {
6
7
8     ListNode *tmp = new ListNode(data);
9
10
11
12     tmp->next = head_;
13
14
15
16     head_ = tmp;
17
18
19
20
21
22 }
```

Linked List: insert(data, index)



Linked List: `_index(index)`

What should the return type of `_index()` be?



[template <class T>]

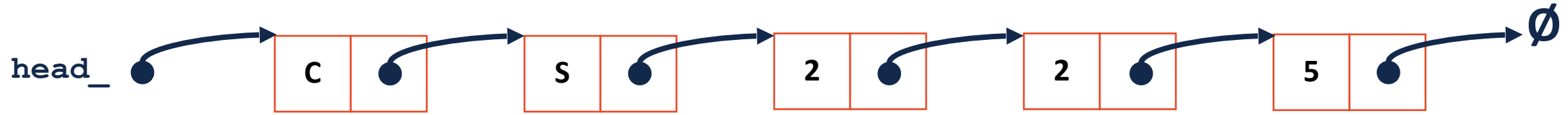
(A) `T &`

(C) `ListNode *`

(B) `ListNode`

(D) `ListNode *&`

Linked List: `_index(index)`



```
58 template <typename T>
59 typename List<T>::ListNode *& List<T>::_index(unsigned index){
60     return _index(index, head_)
61 }
```

```
63 template <typename T>
64 typename List<T>::ListNode *& List<T>::_index(unsigned index, ListNode *& root){
65
66
67
68
69
70
71
72
73 }
```

```
58 template <typename T>
59 typename List<T>::ListNode *& List<T>::_index(unsigned index) {
60     return _index(index, head_)
61 }
```

```
63 template <typename T>
64 typename List<T>::ListNode *& List<T>::_index(unsigned index, ListNode *& root) {
65
66     if (index == 0 || node == nullptr) {
67         return node;
68     }
69
70     return _index(index - 1, root -> next);
71
72
73 }
```

```
1 // Iterative Solution:
2 template <typename T>
3 typename List<T>::ListNode *& List<T>::_index(unsigned index) {
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 }
```

What is the running time for iterative index?

What is the running time for recursive index?



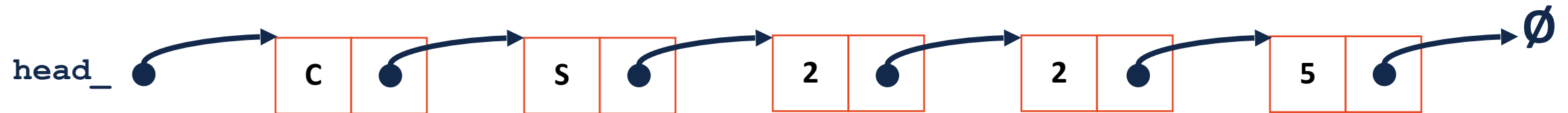
```
1
2 template <typename T>
3 void List<T>::insertAtFront(const T& t)
4 {
5     ListNode *tmp = new ListNode(data);
6
7     tmp->next = head_;
8
9     head_ = tmp;
10
11 }
12
13
14
15
16
17
18
19
20
21
22
```

```
1
2 template <typename T>
3 void List<T>::insert(const T & data,
4 unsigned index) {
5
6
7
8     ListNode *& curr = _index(index);
9
10
11
12     ListNode * tmp = new ListNode(data);
13
14
15
16     tmp->next = curr;
17
18
19
20     curr = tmp;
21 }
22
```

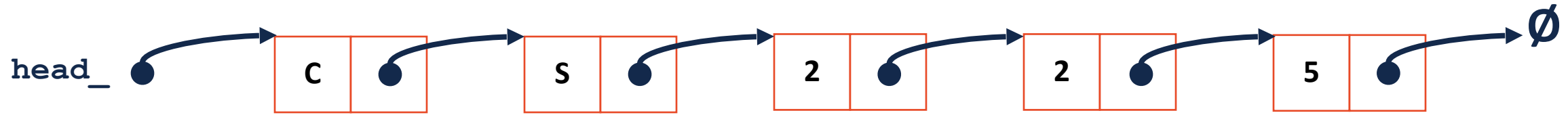
List Random Access []

Given a list L , what operations can we do on L []?

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



Linked List: find(data)



Given a Linked List, what should find return?