

Data Structures

Lists and List ADT

CS 225
Brad Solomon

August 29, 2025



UNIVERSITY OF
ILLINOIS
URBANA - CHAMPAIGN

Department of Computer Science

No class Labor Day (Monday September 1st)

Things to keep in mind for next week:

- Study for exam 0
- Complete POTDs for extra credit
- Set up your own machine!
- Join the CS 225 Discord!
 - Ask questions anonymously in class!
 - Request start of lecture music

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

What is your favorite data structure?

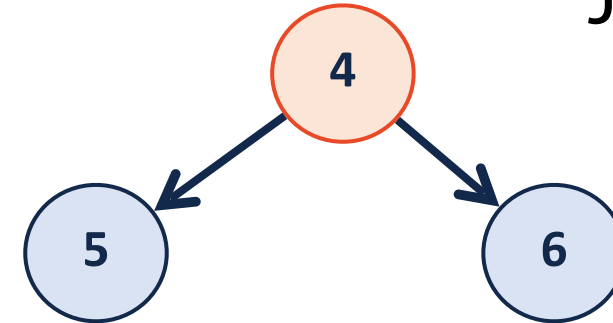


Join Code: 225

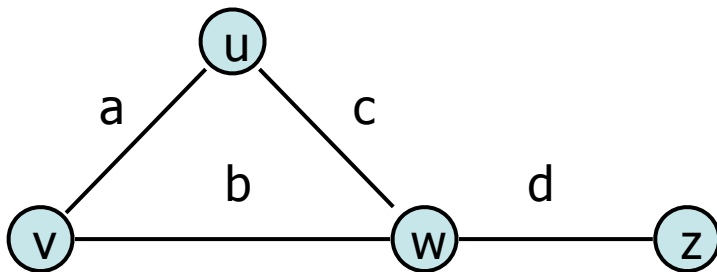
A) Lists



B) Trees



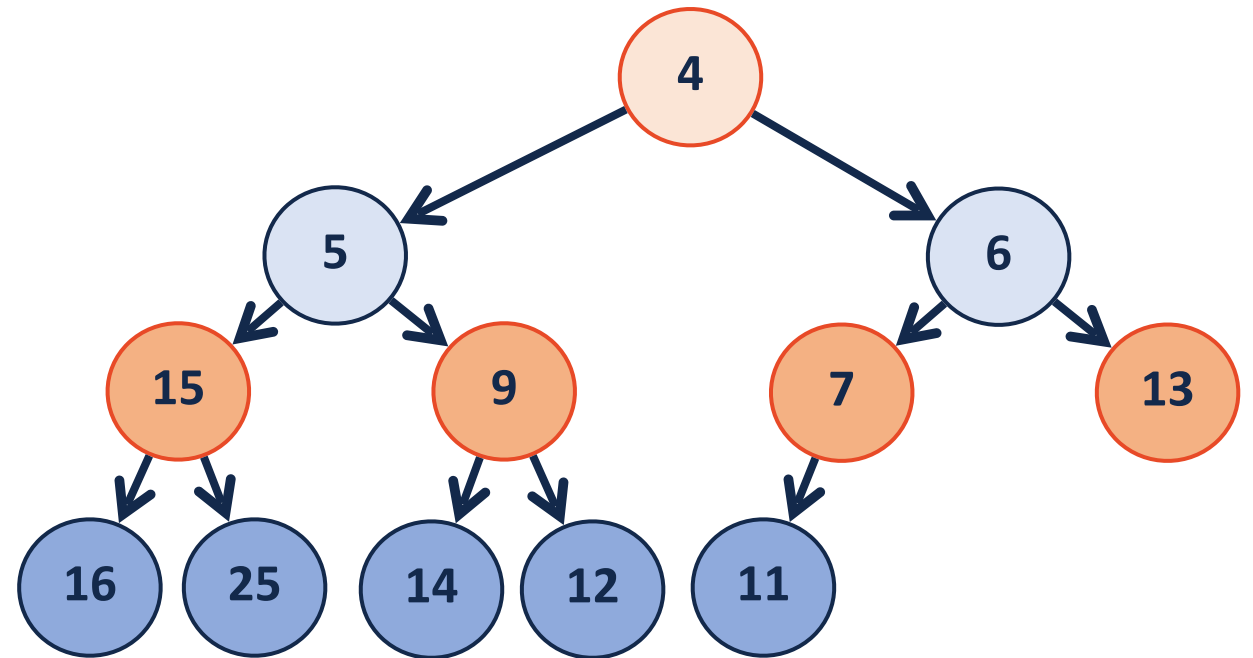
C) Graphs



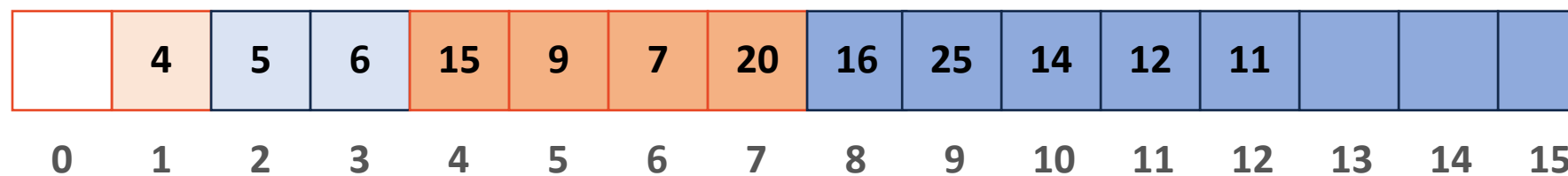
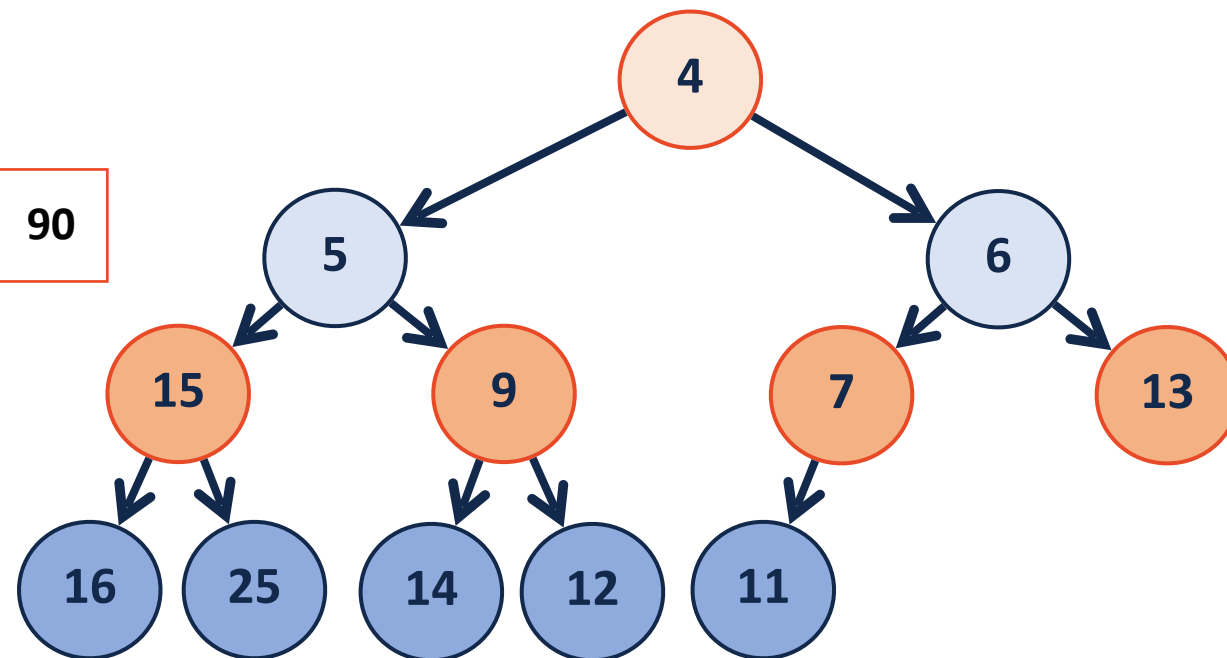
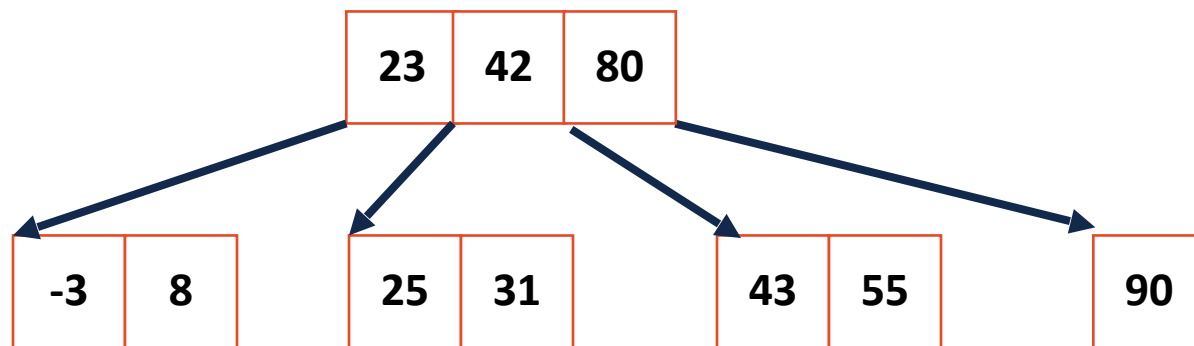
D) Hash Tables

0	Apple
1	∅
2	Pear

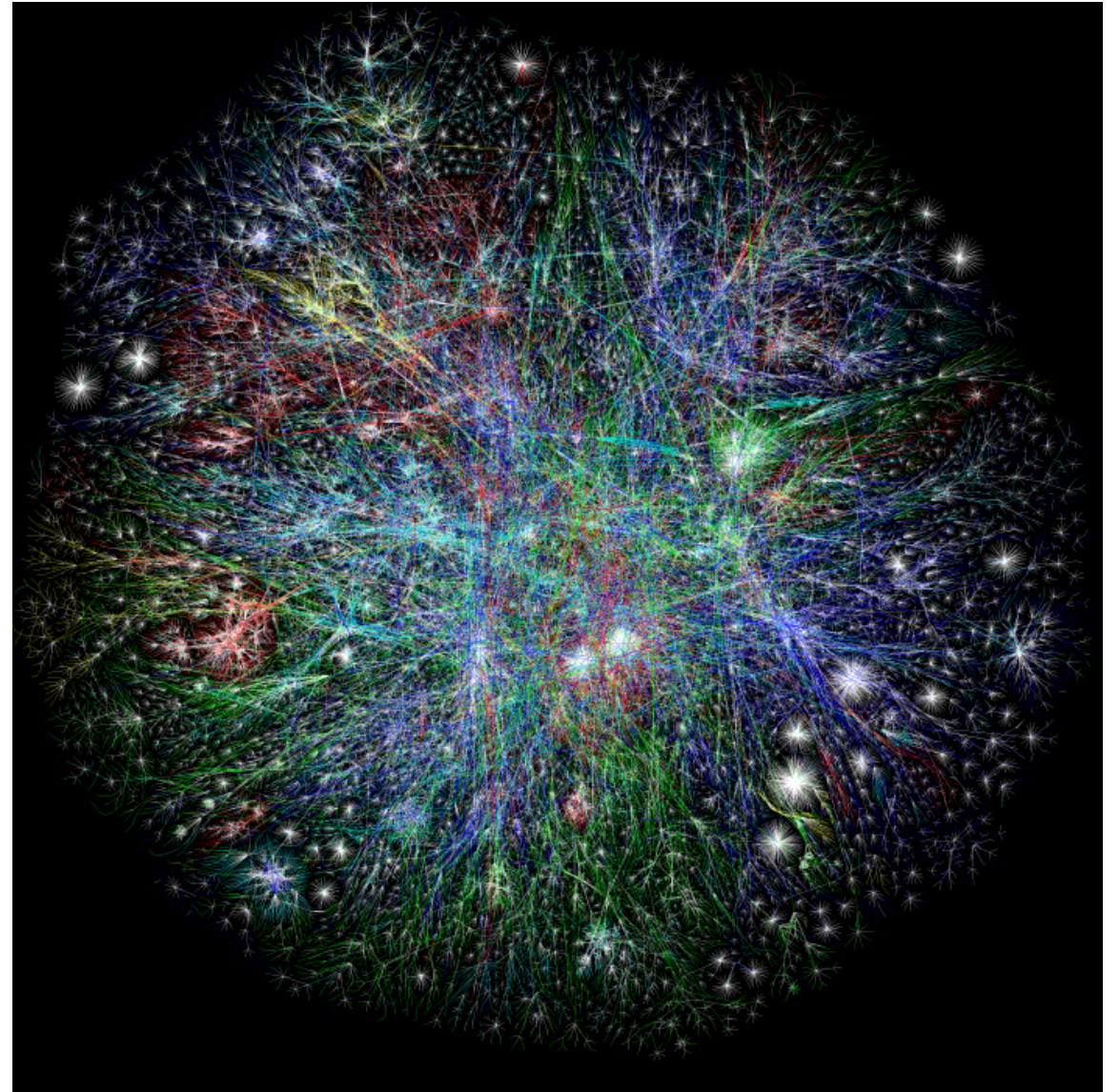
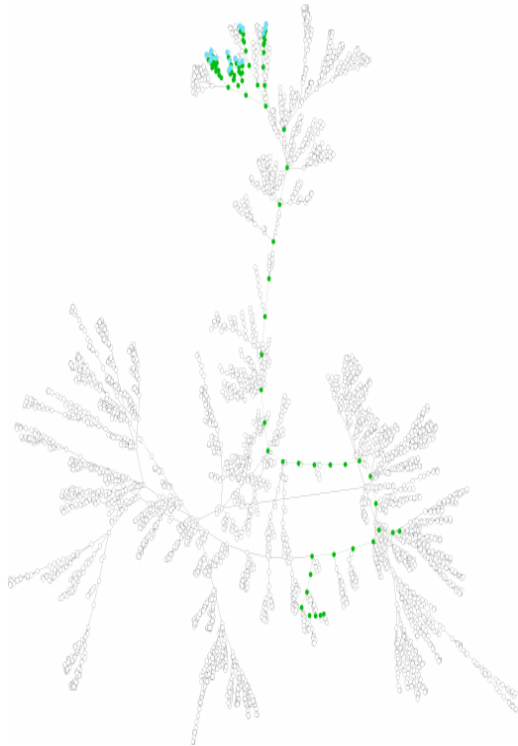
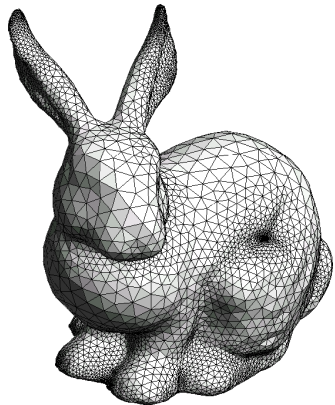
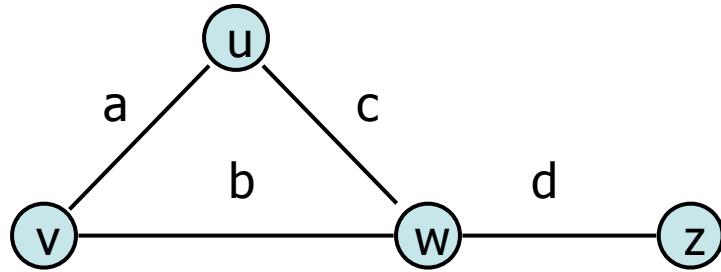
Your favorite data structure: Trees



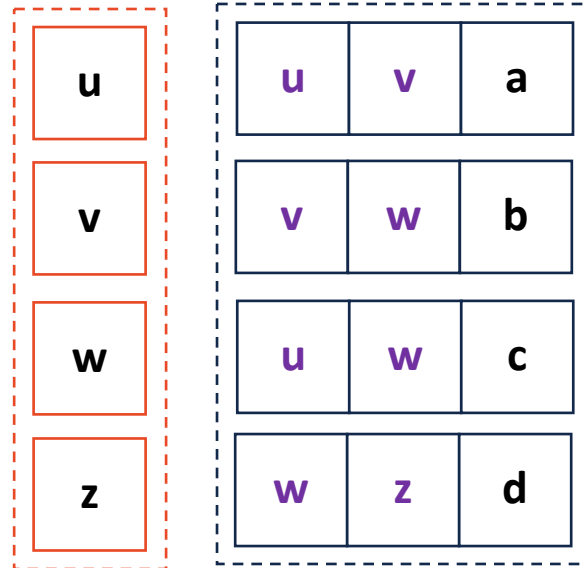
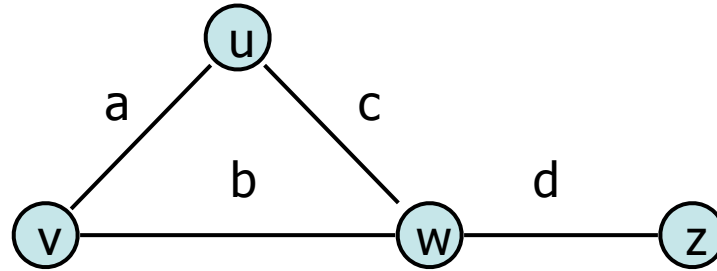
Your favorite data structure: ~~Trees~~ Lists



Your favorite data structure: Graphs

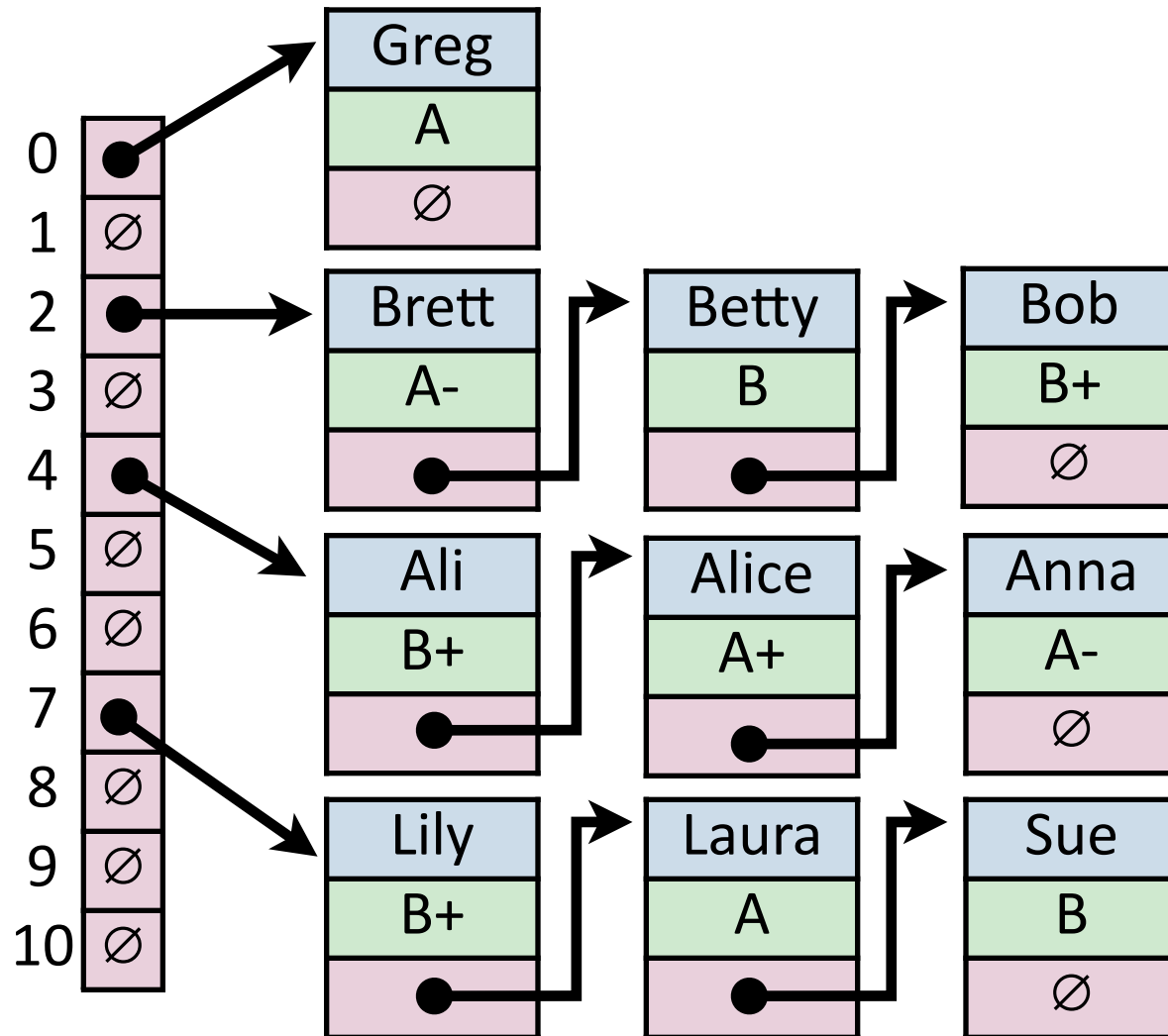


Your favorite data structure: ~~Graphs~~ *Lists*



Your favorite data structure: Hash Tables

$$H = \{h_1, h_2, \dots, h_k\}$$



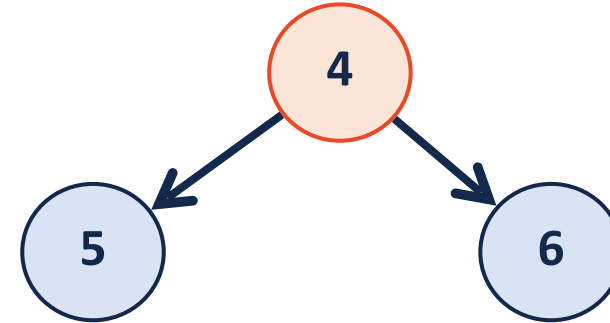
1
0
1
0
1
0
0
1
0
0
0

So 100% of people are excited about lists!

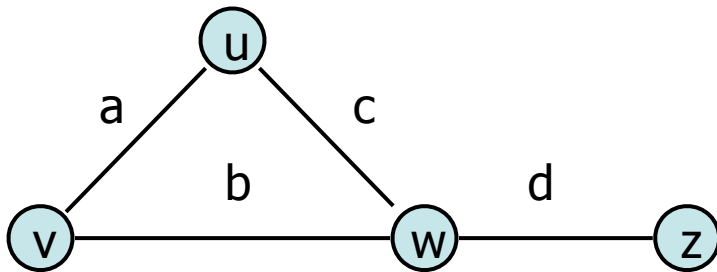
A) Lists



B) Trees



C) Graphs



D) Hash Tables

0	Apple
1	$\emptyset$
2	Pear

Note: Not every tree / graph / hash is actually a list :)

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):

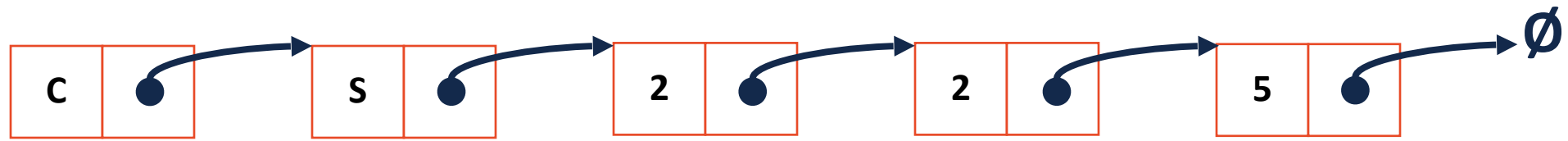
1. Insert
2. Delete
3. isEmpty
4. getData
5. Create an empty list

List Implementations

1.

2.

Linked List



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

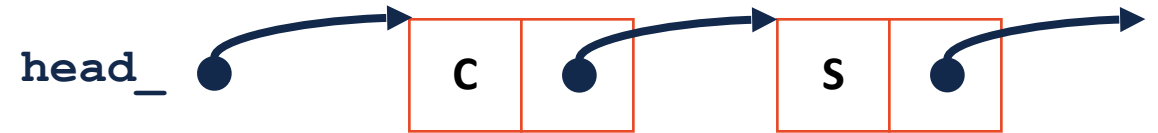
Why is **data** stored as a reference?

Why is **next** a pointer?

List.h

```
1  #pragma once
2
3
4  class List {
5  public:
6      /* ... */
28  void insertAtFront(const T& t);
29
30  private:
31      class ListNode {
32          T & data;
33          ListNode * next;
34          ListNode(T & data) :
35              data(data), next(NULL) { }
36      };
37
38      ListNode *head_;
39
40      /* ... */
...
...  };
79
```

How do I access list given head_?



List.h

```
1  #pragma once
2
3
4  class List {
5      public:
6          /* ... */
28     void insertAtFront(const T& t);
29
30     private:
31         class ListNode {
32             T & data;
33             ListNode * next;
34             ListNode(T & data) :
35                 data(data), next(NULL) { }
36         };
37
38         ListNode *head_;
39
40         /* ... */
...
... };
...
79
```

What is missing in this code?

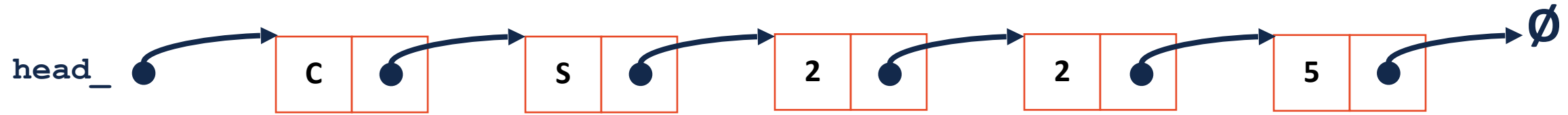
List.h

```
1 #pragma once
2
3 template <typename T>
4 class List {
5     public:
6         /* ... */
28     void insertAtFront(const T& t);
29
30     private:
31         class ListNode {
32             T & data;
33             ListNode * next;
34             ListNode(T & data) :
35                 data(data), next(NULL) { }
36         };
37
38         ListNode *head_;
39
40         /* ... */
...
... };
...
79 #include "List.hpp"
```

List.hpp

```
1
2
3
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

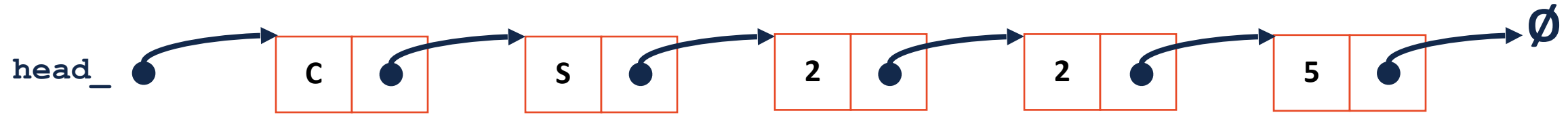
```
1 #pragma once
2
3 template <typename T>
4 class List {
5     public:
6         /* ... */
7
8     private:
9         class ListNode {
10             T & data;
11             ListNode * next;
12             ListNode(T & data) :
13                 data(data), next(NULL) { }
14
15         };
16
17         ListNode *head_;
18
19         /* ... */
20
21 };
22
23 ...
24
25 ...
26
27
28 #include "List.hpp"
```

List.hpp

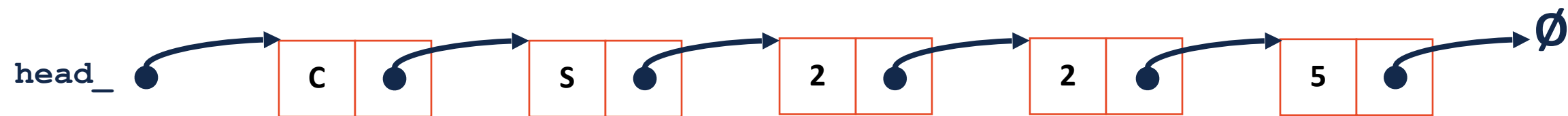


```
1
2
3 template <typename T>
4 void List<T>::insertAtFront(const T& t)
5 {
6
7     ListNode *tmp = new ListNode(data);
8
9
10    tmp->next = head_;
11
12
13    head_ = tmp;
14
15 }
16
17
18
19
20
21
22
```

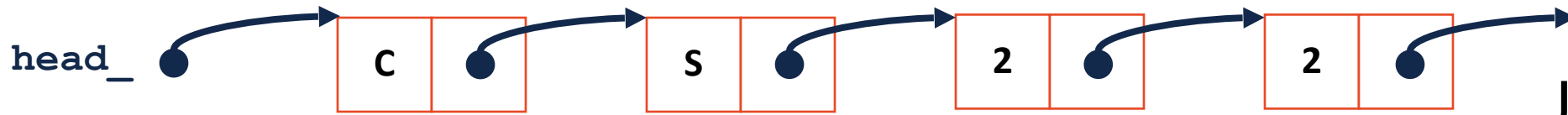
Linked List: insert(data, index)



Linked List: `_index(index)`



Linked List: `_index(index)`



Join Code: 225

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

[template <class T>]

(A) T &

(B) ListNode

(C) ListNode *

(D) ListNode *&


```
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 }
```

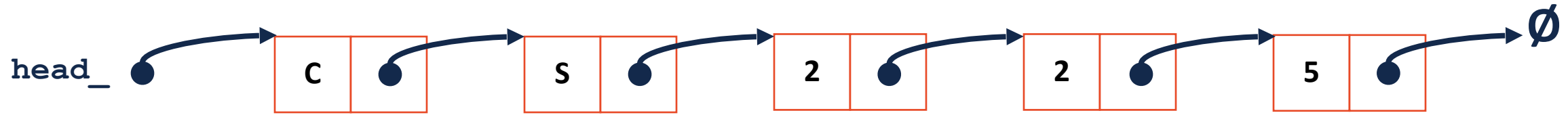
```
58 template <typename T>
59 typename List<T>::ListNode *& List<T>::_index(unsigned index) {
60     return _index(index, head_)
61 }
```

```
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 *curr = head;
7         for (unsigned i = 0; i < index - 1; i++) {
8             curr = curr->next;
9         }
10        return curr->next;
11    }
12 }
```

Which solution is better (iterative or recursive)?

Linked List: insert(data, index)



1) Get reference to previous node's next

```
ListNode *& curr = _index(index);
```

2) Create new ListNode

```
ListNode * tmp = new ListNode(data);
```

3) Update new ListNode's next

```
tmp->next = curr;
```

4) Modify the previous node to point to new ListNode

```
curr = tmp;
```

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

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

Next Time: List Random Access []

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

What return type should this function have?