Programming Abstractions

CS106B

Cynthia Lee

Topics:

- **Link Nodes**
  - LinkNode struct
  - Chains of link nodes
  - LinkNode operations
- **Link Lists**
  - Providing methods for operations on chains of link nodes

Linked Nodes

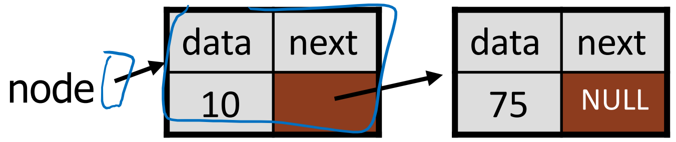
A great way to exercise your pointer understanding

Stanford University

Linked Node

```
struct LinkNode {
   int data;
   LinkNode *next;
}
```

- We can chain these together in memory:



node->next->data = 75;

```
LinkNode *node1 = new LinkNode;      // complete the code to make picture
node1->data = 10;
node1->next = NULL;
LinkNode *node = new LinkNode;
node->data = 10;
node->next = node1;
```

FIRST RULE OF LINKED NODE/LISTS CLUB:

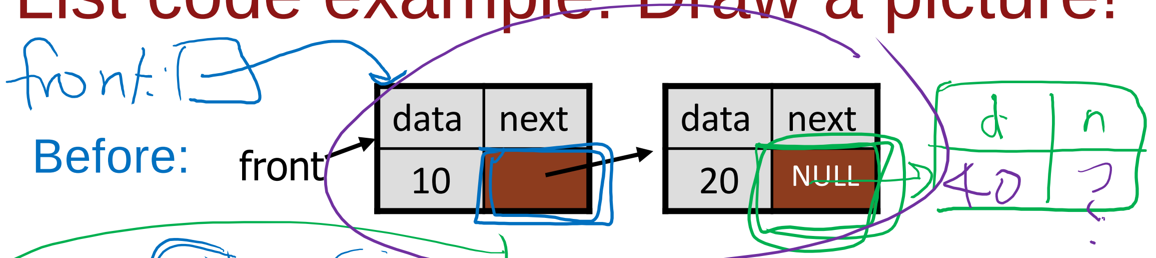
DRAW A PICTURE OF LINKED LISTS

Do no attempt to code linked nodes/lists without pictures!

List code example: Draw a picture!

```
struct LinkNode {
   int data;
   LinkNode *next;
}
```

Before:

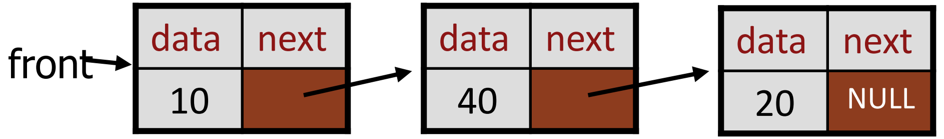


```
front->next->next = new LinkNode;
front->next->next->data = 40;
```

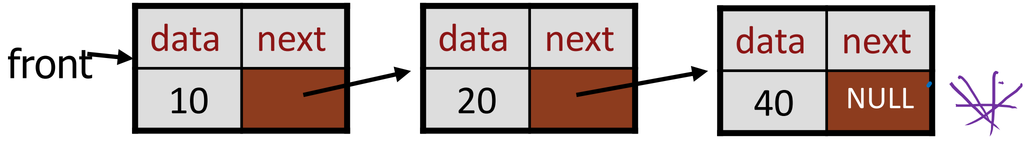
front -> next -> next -> next = NULL;

LinkNode * n = new LinkNode;

A. After:



B. After:



C. Using "next" that is NULL gives error

D. Other/none/more than one

Stanford University

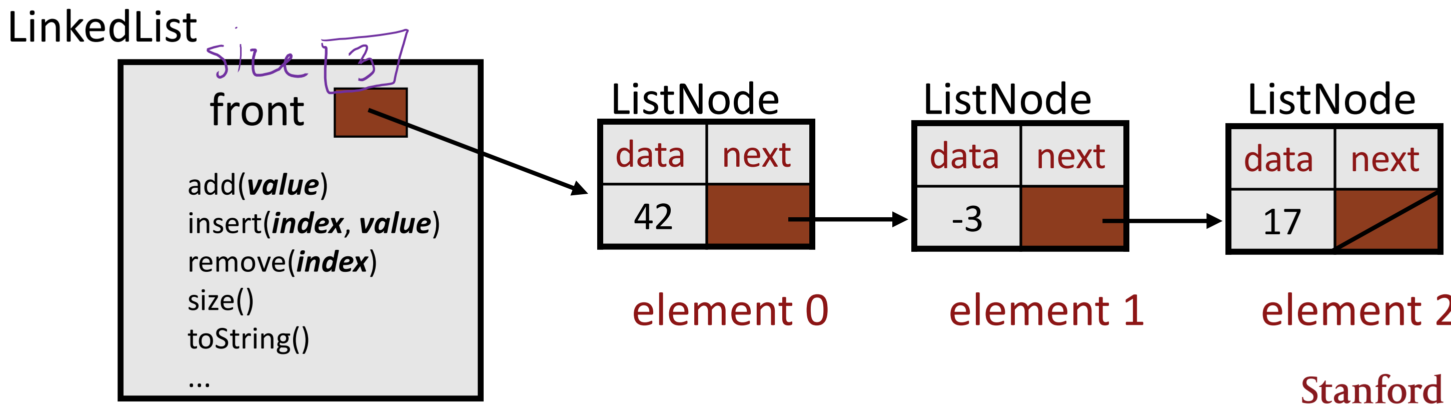
Linked List Data Structure

Putting the ListNode to use

A LinkedList class

Let's write a collection class named `LinkedList`.

- Has the same public members as `ArrayList`, `Vector`, etc.
  - `add`, `clear`, `get`, `insert`, `isEmpty`, `remove`, `size`, `toString`
- The list is internally implemented as a **chain of linked nodes**
  - The `LinkedList` keeps a pointer to its `front` node as a field
  - `NULL` is the end of the list; a `NULL` front signifies an empty list

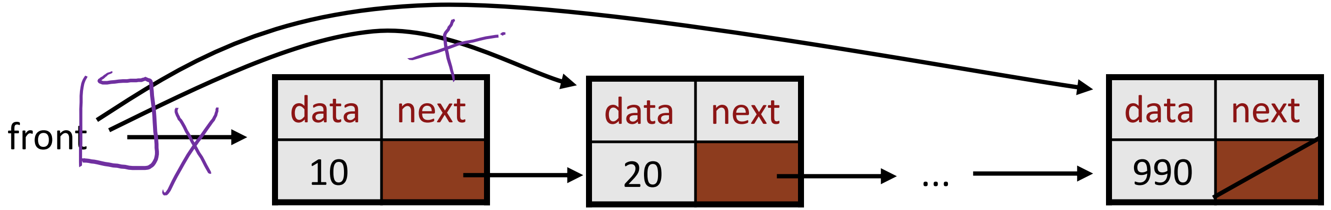


Traversing a list? (BUG version)

What's wrong with this approach to traverse and print the list?

```
while (front != NULL) {
    cout << front->data << endl;
    front = front->next;    // move to next node
}
```

- *It loses the linked list as it is printing it!*

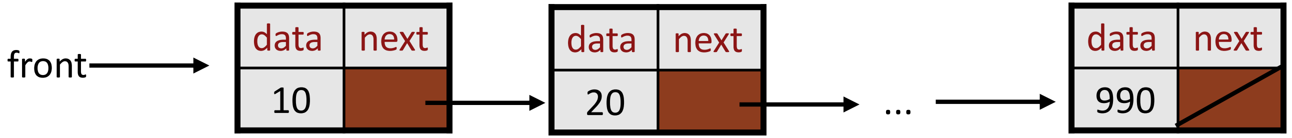


Traversing a list (12.2) (bug fixed version)

The correct way to print every value in the list:

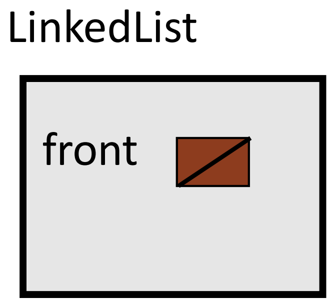
```
ListNode* current = front;
while (current != NULL) {
    cout << current->data << endl;
    current = current->next;  // move to next node
}
```

- Changing `current` does not damage the list.



LinkedList.h

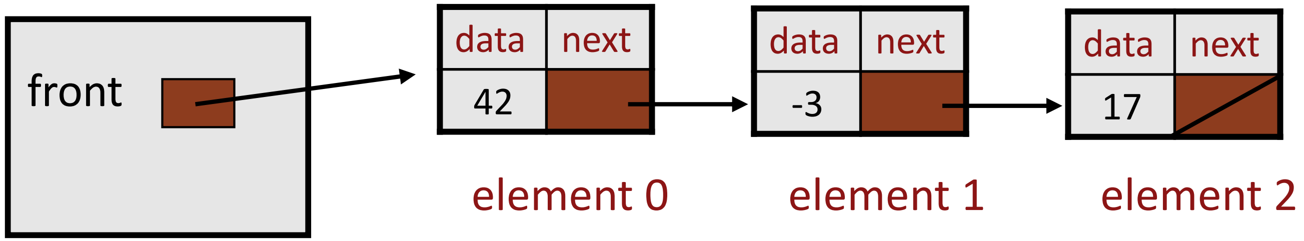
```
class LinkedList {
public:
    LinkedList();
    ~LinkedList();
    void add(int value);
    void clear();
    int get(int index) const;
    void insert(int index, int value);
    bool isEmpty() const;
    void remove(int index);
    void set(int index, int value);
    int size() const;

private:
    ListNode* front;
    int size;
};
```



Implementing add

```
// Appends the given value to the end of the list.
void LinkedList::add(int value) {
    ...
}
```

- What pointer(s) must be changed to add a node to the end of a list?
- What different cases must we consider?


front
data
next
42
data
next
-3
data
next
17
element 0
element 1
element 2