SINGLY LINKED LISTS

CS A250 – C++ Programming II

INTRODUCTION

- **Singly-linked list**
  - Constructed using **pointers**
  - *Grows* and *shrinks* during __runtime__
  - **Doubly-linked lists**:
    - A variation with pointers in both directions
- **Pointers** are the backbone of such structures
  - Use *dynamic* variables
- **Standard Template Library**
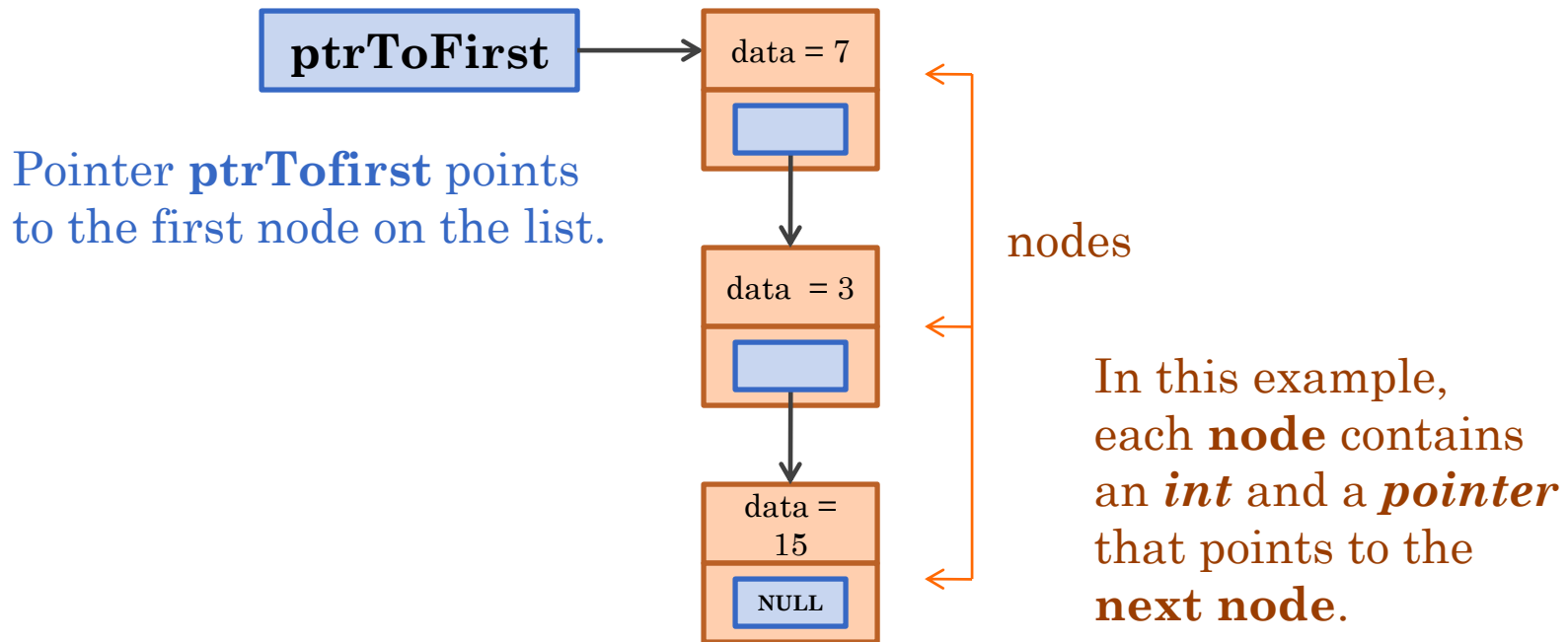 - Has predefined versions of linked lists

APPROACHES

- Three ways to handle such data structures:
    1. **C-style approach:** global functions and **structures** with everything **<u>public</u>**

    2. **Classes** with **<u>private</u> member variables** and **accessor** and **mutator** functions

    3. **Friend classes**

- We will use approach 2

3

NODES AND LINKED LISTS

- **Linked list**
  - Simple example of "dynamic data structure"
  - Composed of **nodes**
- Each "**node**" is a **variable** of **class type** that is *dynamically* created with **new**
  - Nodes also contain **pointers** to other nodes.

NODES AND POINTERS

ptrToFirst

data = 7

Pointer **ptrTofirst** points to the first node on the list.

data = 3

nodes

data = 15

NULL

In this example, each **node** contains an ***int*** and a ***pointer*** that points to the **next node**.

5

LINKED LIST

- Lists as illustrated are called **linked lists**
- **First node** pointed to by pointer usually named **head** or **first**
  - We will call it **ptrToFirst** for now
- **Last node** is the *end* marker
  - Always set the pointer of last node to `NULL`
    - Considered "sentinel" because it indicates no further "links" after this node
    - Easy to test for "end" of linked list.

6

LINKED LIST IMPLEMENTATION

- To implement a **linked list** we need **2 classes**:
  - A **class** to create a **single node**
  - A **class** to create a **list composed of nodes**
- For our example, we will have nodes that have *only* two pieces of data:
  - An **integer**
  - A **pointer** that links to another node

LINKED LIST IMPLEMENTATION (CONT.)

- **Node class**
  - Creates a **node** that has two member variables (can have more):
    - An **integer** name **data** storing some number
    - A **pointer** named **ptrToNext** that we set to point to the next node
      - This **pointer** is usually named **next** or **link**
- We will have all **member functions definitions** *inline*
  - Because the class is short and simple enough.

Node Class Definition

```cpp
class Node
{
public:
    Node() : data(0), ptrToNext(NULL){}
    Node(int newData, Node *newPtrToNext)
        : data(newData), ptrToNext(newPtrToNext){}
    Node* getPtrToNext( ) const { return ptrToNext; }
    int getData( ) const { return data; }
    void setPtrToNext( Node * newPtrToNext )
        { ptrToNext = newPtrToNext; }
    void setData( int newData ) { data = newData; }
    ~Node(){}
private:
    int data;
    Node *ptrToNext;
};
```

ANYLIST CLASS

- Once we have the **Node class**, we need a class that creates a **list** composed of **nodes**

- In our example, we implement a class named **AnyList**
  - Creates a **list** with
    - A **pointer ptrToFirst** that points to the *first* **node** of the **list**
    - A **counter count** to keep track of how many **nodes** are in the **list**.

10

EXAMPLE

- **Project:** 01_singly_linked_lists
  - AnyList.h

HOW TO CREATE THE FIRST NODE

- **`Node *ptrToNode;`**
  - Creates a *pointer* to point to a **new** node
- **`ptrToNode = new Node;`**
  - Creates a **new** *node*
- **`ptrToNode->setData(3);`**
  - Stores 3 in the member var **data** (assuming we have an *int*)
- **`ptrToNode->setPtrToNext(NULL);`**
  - Pointer in new node set to NULL (since it is the only node)
- **`ptrToFirst = ptrToNode;`**
  - Set *pointer* **ptrToFirst** to point to the new node

- Operator **`->`**
  - Called **arrow operator**
  - Shorthand notation that combines " ***** " and " **.** "

EXAMPLE

- **Project:** 01_singly_linked_lists
  - **Constructor**
  - How to create a node

- **NOTE:** *Since one of the member variables of the **SinglyLinked List** class is dynamic, the class should include a **copy constructor** and an **overloaded assignment operator**. For practical purposes, we will omit these until we address these topics later in the semester.*
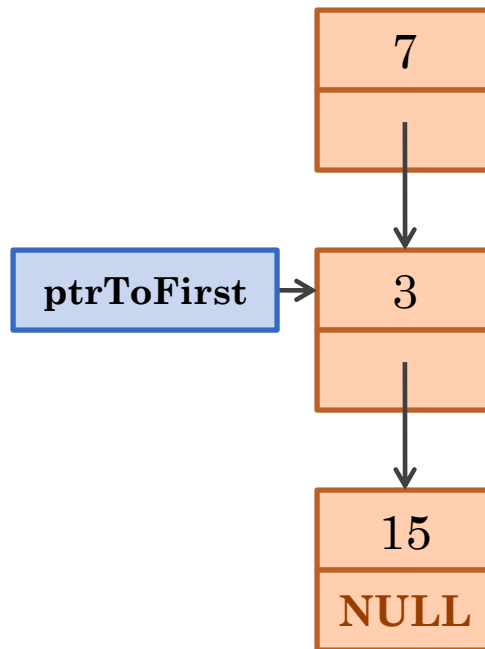
13

INSERTING TO THE FRONT OF THE LIST

○ To **insert a node** to the *front* of the list,

you need to:

1. Create a pointer to point to a **new** node
2. (this is *dynamic*)
3. Create a new node
4. Store data in the new node
5. Set **new node's pointer** to point to the **first node**
6. Make the new node be the "first" node
7. Increment the count

○ **Note:** If the **list** is **empty**
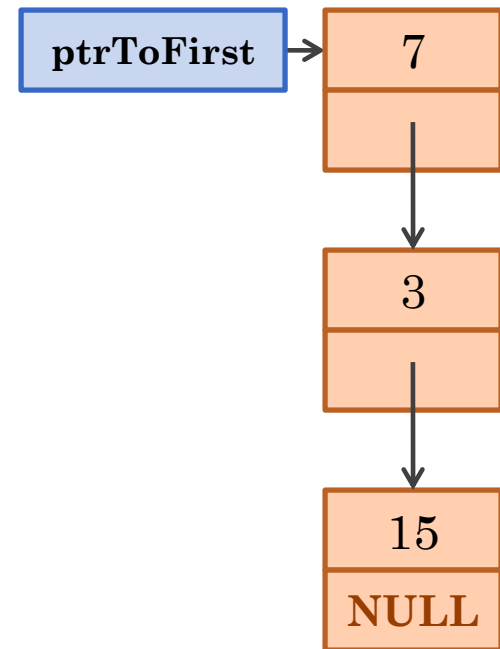 • Then the new node is the **first** and **only** node.

14

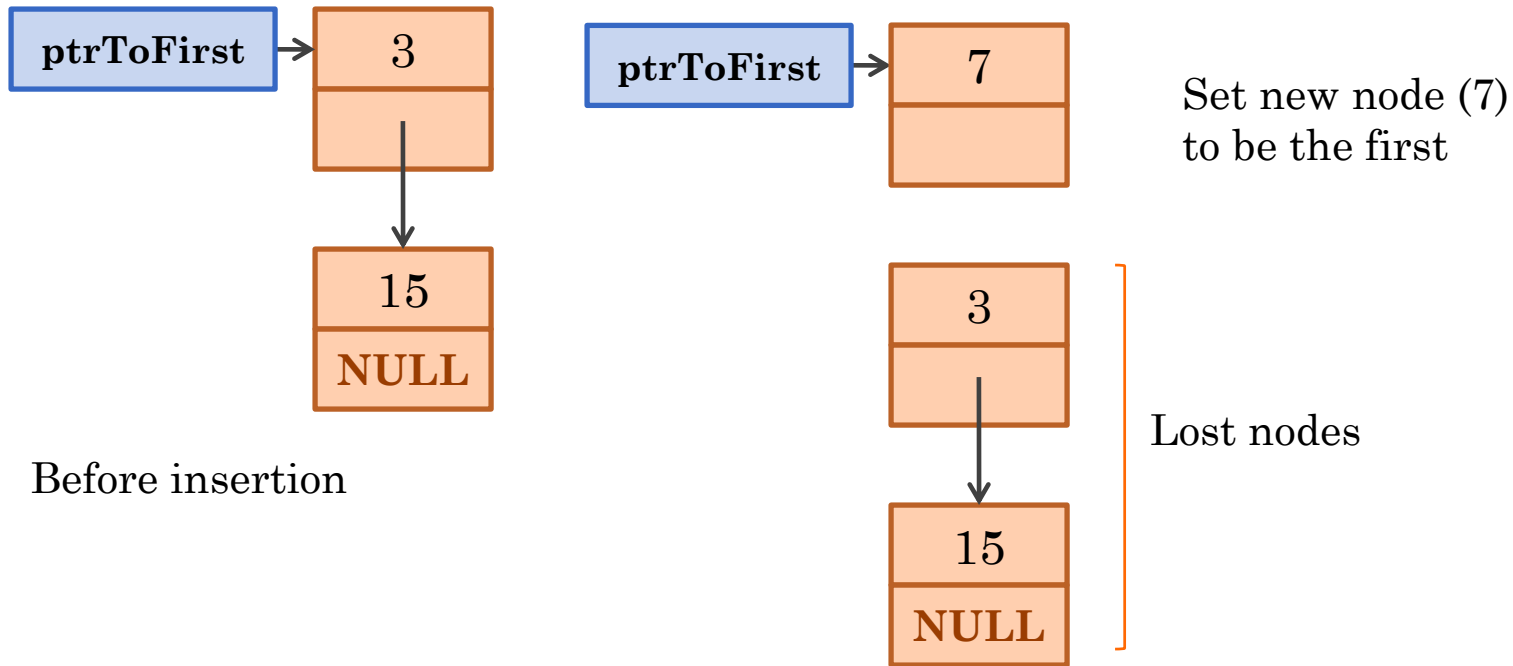# INSERTING TO THE FRONT OF THE LIST
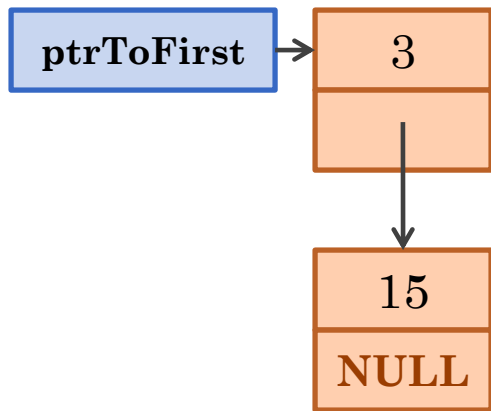
Before insertion

Insert new node (7)

Make *ptrToFirst*
point to the new node

15

PITFALL: LOST NODES

| ptrToFirst | → | 3 |
| | | |

| 15 |
| NULL |

Before insertion

| ptrToFirst | → | 7 |
| | | |

Set new node (7)
to be the first

| 3 |
| |

| 15 |
| NULL |

Lost nodes

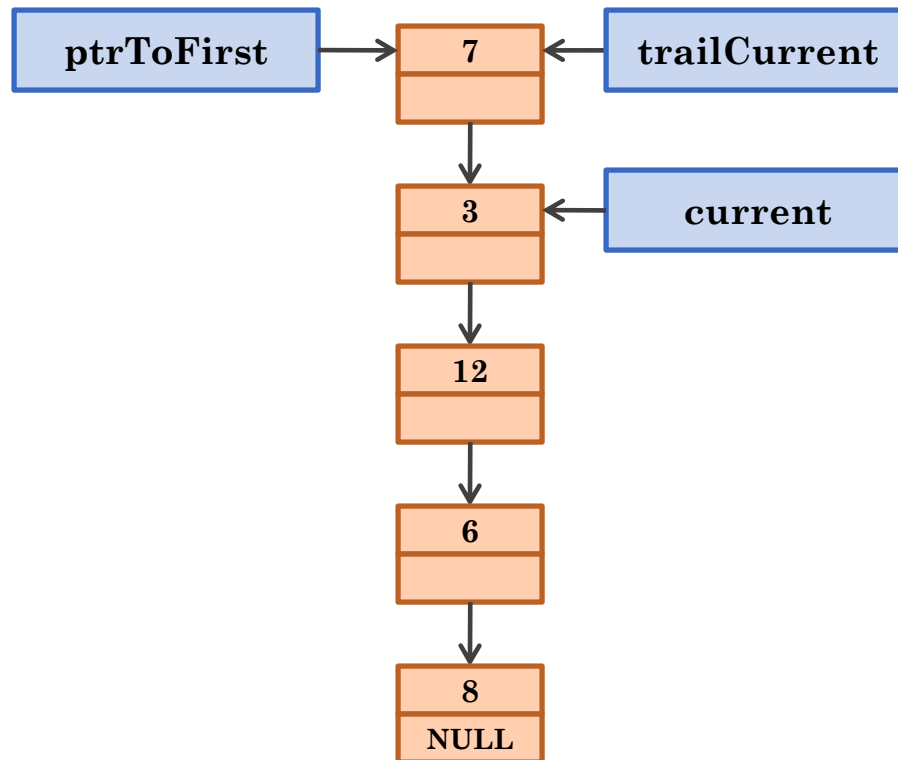# EXAMPLE

- **Project:** 01_singly_linked_lists
  - Function: **insertFront( )**
 - Inserting to the front of the list

REMOVING A NODE

- We want to delete a node that has a given information. What do we need?
  - Create a **pointer** to
    - traverse the list → `current`
    - be right behind `current` → `trailCurrent`
  - Set a **boolean** value to keep track of whether the item is found or not
- Need to consider *all* cases:
  - List is empty → output message
  - Node to be deleted is first
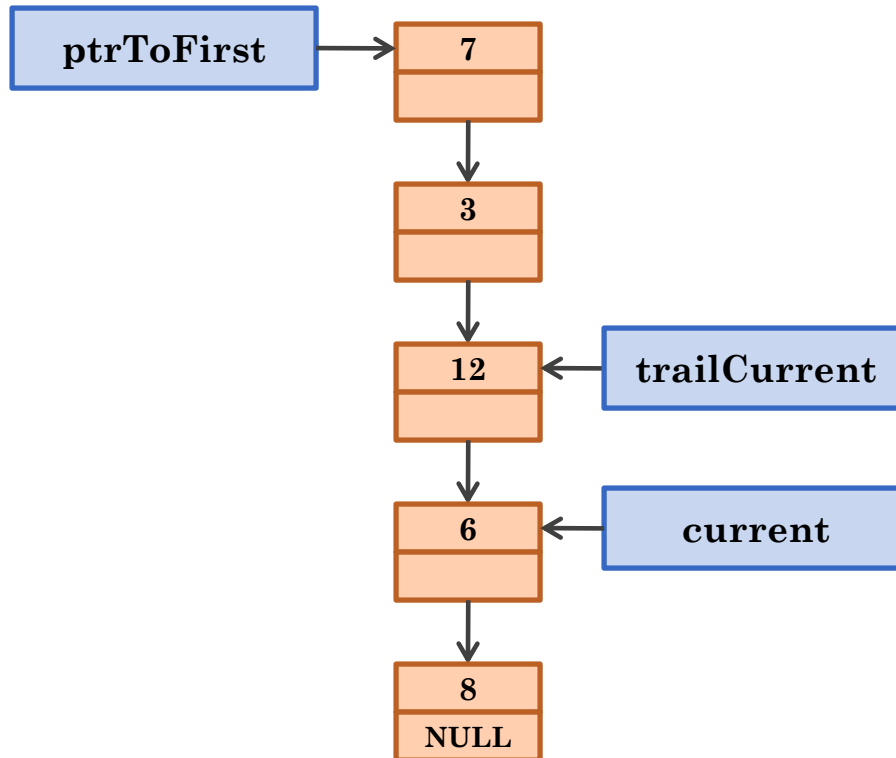 - Item was not found

DELETING A NODE

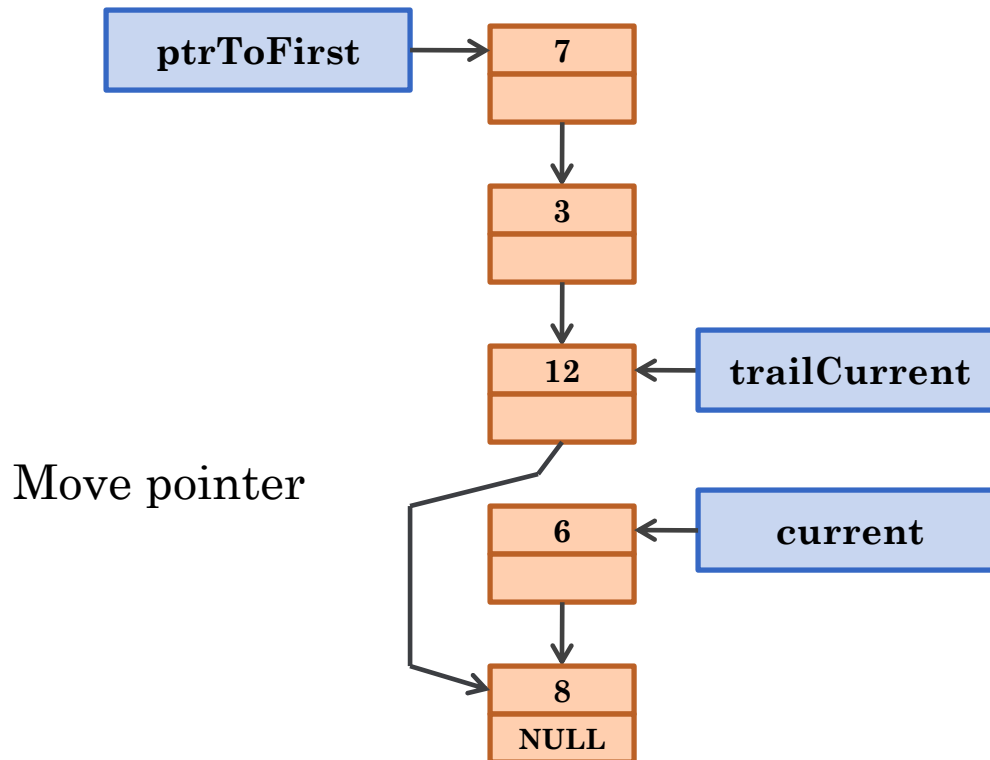Looking for (6)

DELETING A NODE (CONT.)

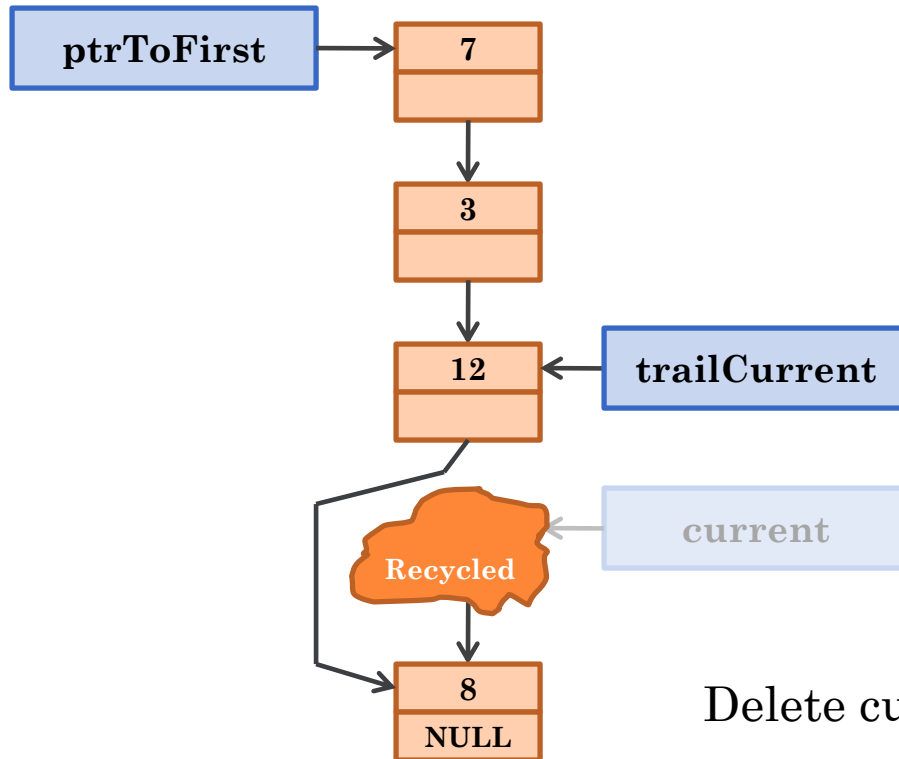Looking for (6)

DELETING A NODE (CONT.)

ptrToFirst → 7

3

12 ← trailCurrent

6 ← current

8
NULL
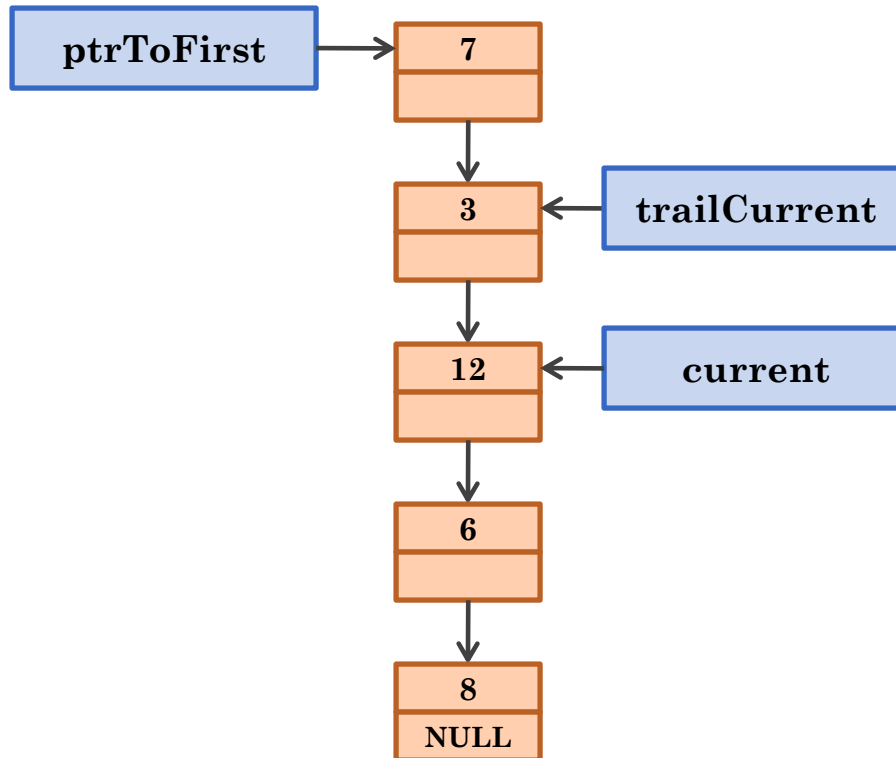
Looking for (6)

Found!

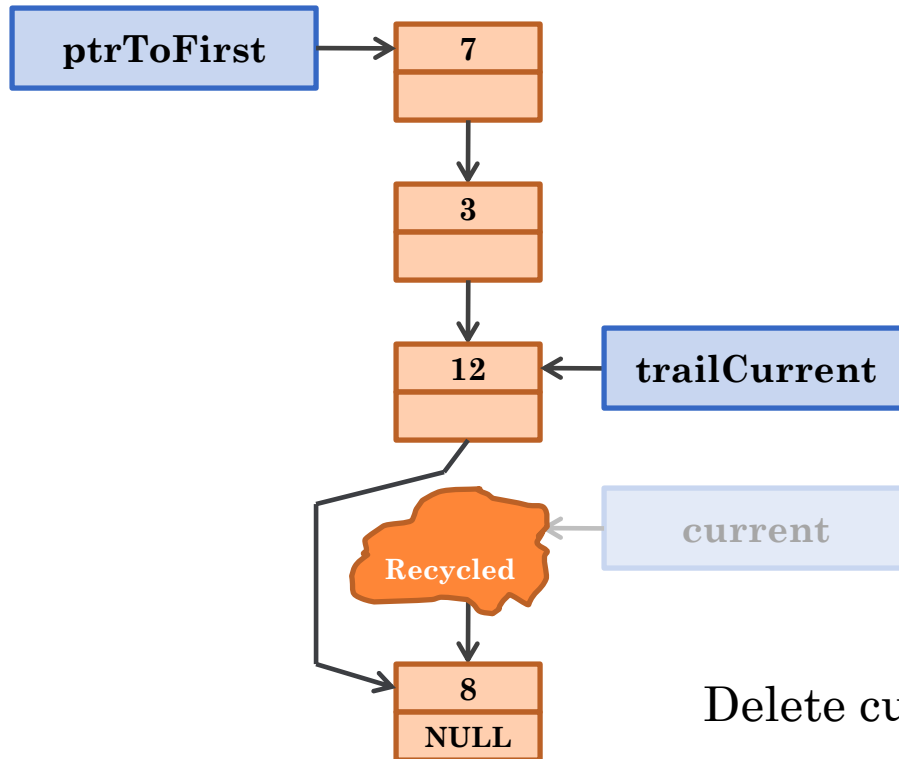# Deleting a Node (cont.)

Looking for (6)

Move pointer

DELETING A NODE (CONT.)

Looking for (6)

Delete current

DELETING A NODE (CONT.)

Looking for (6)

Delete current

EXAMPLE

- **Project:** 01_singly_linked_lists
  - Function: **deleteNode()**
  - Deleting a node
    - Need to consider all cases
      - List is empty
      - Node to be deleted is the first
      - Continue searching for node in the list
      - Node is found
      - Node is not found

PRINTING THE LIST

○ How do you **print** the list?

1. Create a **pointer** to traverse the list → **current**

2. Set the **current** pointer to point to the **first** node

3. While the **current** pointer does not point to **NULL** (that is, has not reached the end of the list)

   a) Output the data the **current** pointer is pointing to

   b) Move the **current** pointer forward

EXAMPLE

- **Project:** 01_singly_linked_lists
  - Function: **print()**

LIST DESTRUCTOR

- The **destructor** will call a function *destroyList( )* to perform all necessary operations to free memory

  1. Create a pointer **temp** to traverse the list

  2. Use a **while loop** to delete all nodes in the list

  3. Reset **count** to 0

28

USEFUL SYNTAX

- You have several nodes (*n1*, *n2*, *n3*…)
  - *n1* is the **first** node
- You want to…
  - … Get the data stored in node *n1*

    ```
    ptrToFirst->getData();
    ```

  - … Make a new node be the be first and point to *n1* (assume you already created **ptrToNewNode**)

    ```
    ptrToNewNode->setPtrToNext(ptrToFirst); //point to n1
    ptrToFirst = ptrToNewNode; //the new node is now
                              // the first node
    ```

  - … Know if the list is empty

    ```
    if (ptrToFirst == NULL)
    ```

29

COMMON ERRORS

- Forgetting to add
  - **#include** **<string>** in the **AnyList.h** file
    - Needed for **NULL**

- Confusing nodes and pointers

- Forgetting to reset the pointer that points to the first node in the list, *ptrToFirst*

IMPORTANT !

- **Before executing** your program (F5) *always* do the following:

  - Click on **Build** → **Rebuild Solution**

IMPORTANT → COMMON IDENTIFIERS

- We have named the pointer that points to the first node **ptrToFirstNode**

  - BUT, common identifiers are:**first**, **head**

- We have named the pointer that points to the next node **ptrToNextNode**

  - BUT, common identifiers are: **link**, **next**

- We have named the pointer that points to a new node **ptrToNewNode**

  - BUT, most common identifier is: **newNode**

32

IMPORTANT → COMMON IDENTIFIERS

- We have named the pointer that points to the first node **ptrToFirstNode**

  - BUT, common identifiers are: **first**, **head**

- We have named the pointer that points to **ptrToNextNode**

  - BUT, common identifiers are: **link**, **next**

Class projects
may use
<u>any</u> identifier.

- We have named the pointer that points to a new node **ptrToNewNode**

  - BUT, most common identifier is: **newNode**

END SINGLY-LINKED LISTS

34