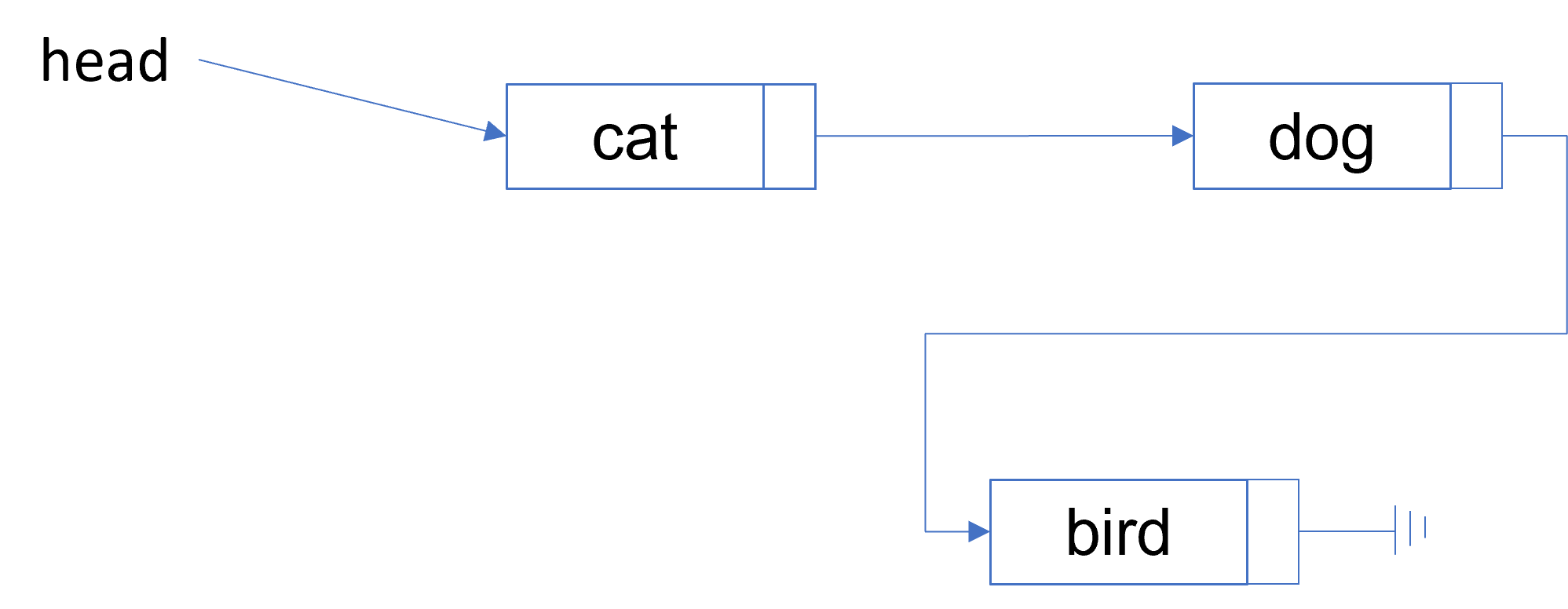
# Lecture Notes, week of Tues May 9

## Example 1

Make this linked-list data structure in C++:



Please see **singly\_linked\_list0.cpp** for a walk-through of the creation of this data structure. It explains most of the basic ideas of a linked list in a concrete way.

A few general notes … each “node” of a linked-list stores at least two things: it’s data value, and a pointer to the next node in the list. So we can represent a node like this:

struct Node {  
 string data;  
 Node\* next;  
};

The node stores a string, plus a pointer to the next node in the list. The next pointer for the last node of the list will have the value nullptr.

We need to know what node is at the start of the list, so will use a special Node pointer for that:

Node\* head = nullptr;

head always points to the first element of the list. When head is nullptr, the list is empty.

To create a new node for the list, we do this:

head = new Node{"hello", nullptr};

This creates a new Node, and also makes head point to it. This is now a linked list of length 1.

To delete the node, we can do this:

delete head;  
head = nullptr;

**Remember**: Every time we call new, there must, eventually, be a call to delete that de-allocates the new-ed memory. Otherwise you have a **memory leak**.

## Example 2

Let’s generalize the ideas of Example 1 by writing functions that answer the following questions (see **singly\_linked\_list1.cpp**):

How do you test if the list is **empty**?

How do you add a new node to the **front** of the list?

How do you remove the node at the **front** of the list?

How do you count the number of nodes on the list?

How do you print the nodes on a list?

How do you remove all elements?

## Example 3

Now lets re-write the functions from Example 2 in a class called List (see **singly\_linked\_list2.cpp**). But encapsulating all the code for the list into a class we make it much easier for an application to use: they don’t need to worry about pointers, or Nodes, or allocation and de-allocating memory.