Linked Lists

- Lists are containers that store a collection of data, all of the same type.

  e.g. A list of point objects may be used to define a polygon in a graphics program.

- Lists are generic access containers with the following methods:

+ INSERT (object) :	adds object to List
+ DELETE (Key) :	removes object with matching Key from List
+ RETRIEVE(Key) :	returns a copy of object in List with matching Key.

Linked Lists

- Notes on special conditions:
  + If the List stores unique objects, INSERT only adds the object if not already contained.
  + If the List does not contain the object to DELETE, the method must gracefully return.
  + If the List does not contain the object to RETRIEVE, the method must return a default constructed object.

Linked Lists

- Lists may be implemented with a variety of internal structures, as long as the methods are correctly implemented.
  common structures:

  + Array
    Sequence of objects stored in sequential memory. Updates may require movement of large numbers of objects, or allocation of new array space and movement of all objects.
    Allows reference to objects' locations by array indexing.

  + Linked Nodes
    Each object stored in separate Node with link to next node. Updates and retrieval require traversal of nodes to find object. Never need to move data.

Question:

If each Node points to the next Node in the List, how is the last Node identified?

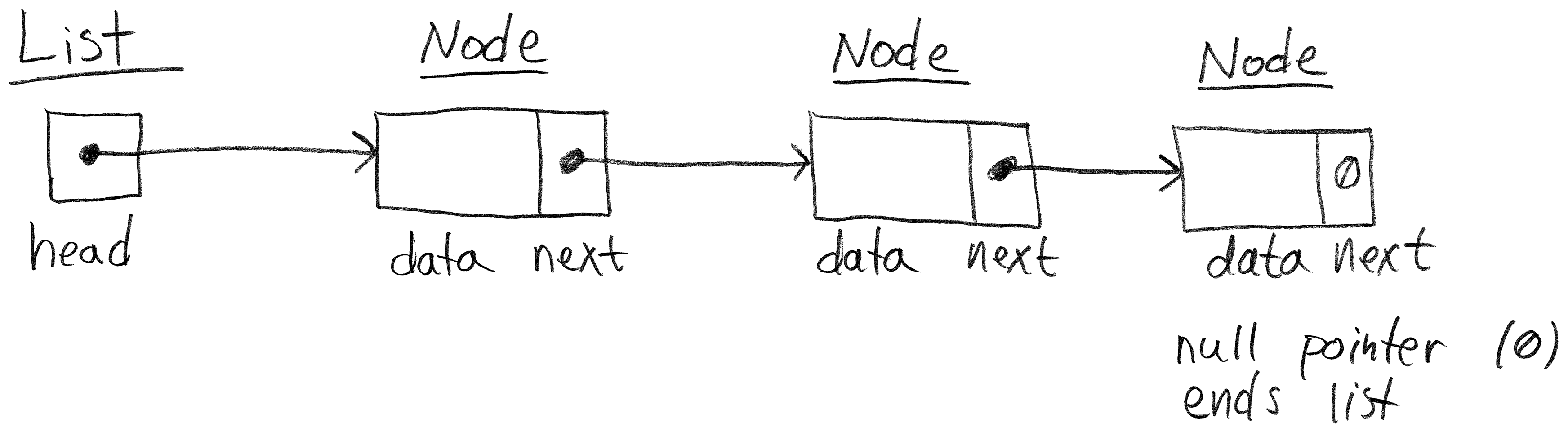
Question:

If each Node points to the next Node in the List, how is the first Node identified?

Question:

How is an empty List represented?

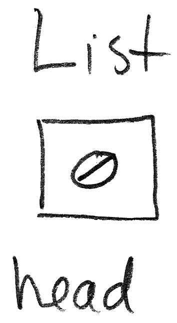
Linked Lists



```
class Node
{
 public:
    DataType data;
    Node  *next;
};
```

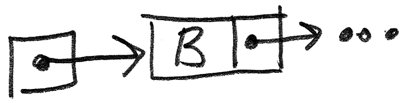
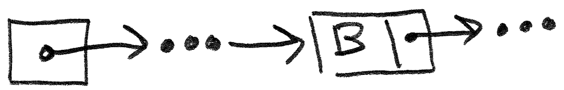
```
class List
{
 public:
    :
 protected:
    Node *head;

};
```

Linked Lists



Empty List

Example of RETRIEVE(B):

Case	Before
First Node	
Not First Node	

What are the steps?

Write pseudo-code on paper.

What if B isn't stored?

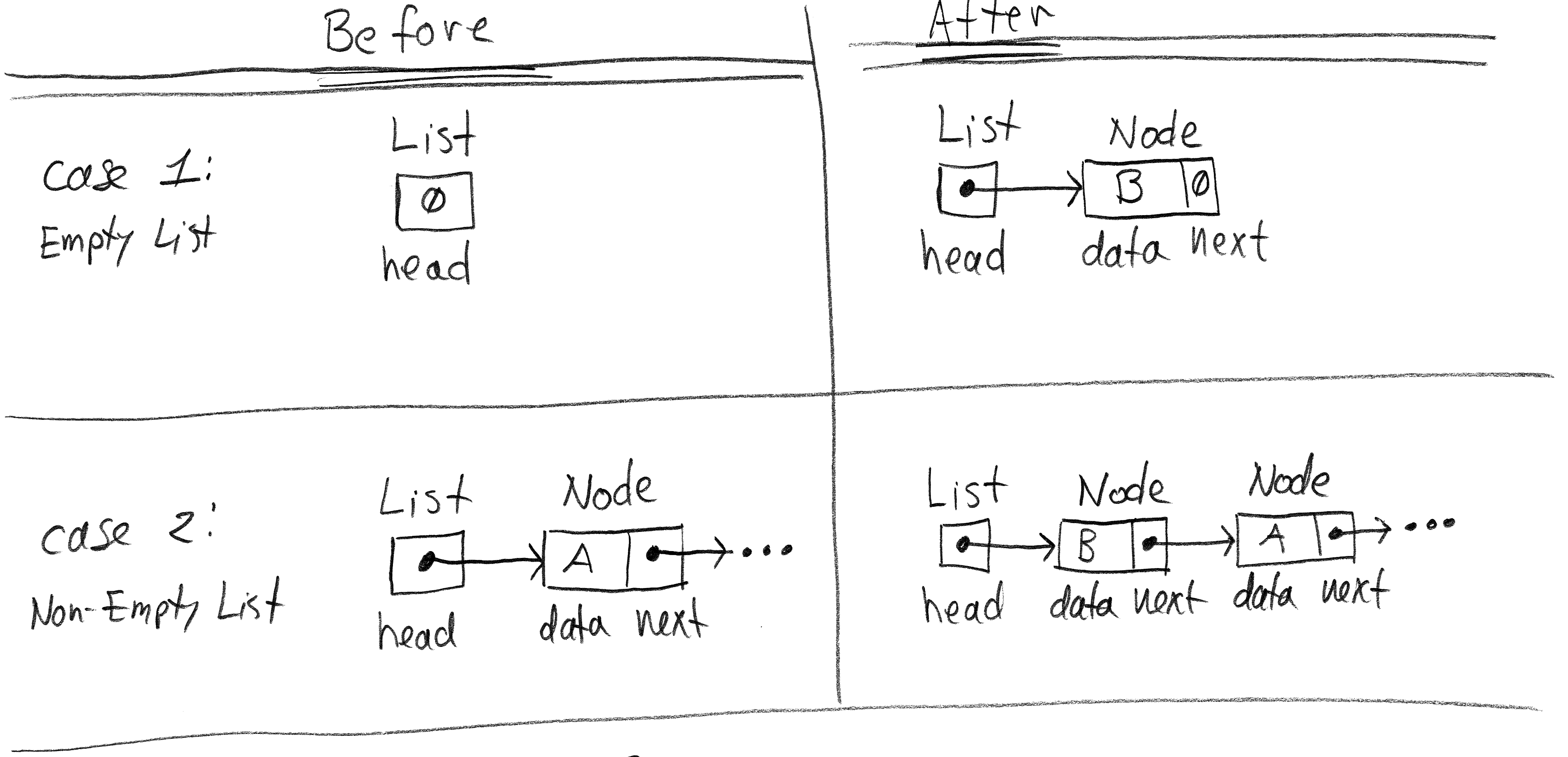
Does it cover all cases?

```
RETRIEVE(Key):
    Set current Node to head Node
    until end is found:
        if current's data matches Key:
            return current's data
        set current to current's next
    return default object of data's type
```

Linked Lists

Insert

Unordered Lists don't care about position. Choose to insert at front. Example of INSERT(B):

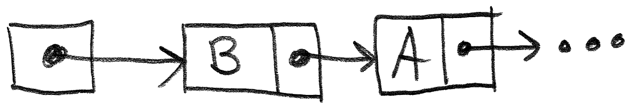
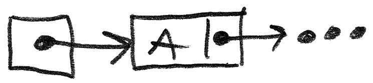
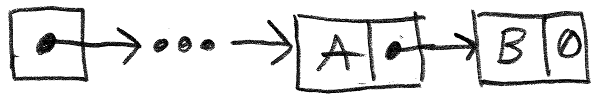
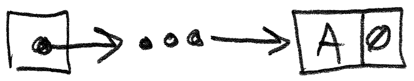
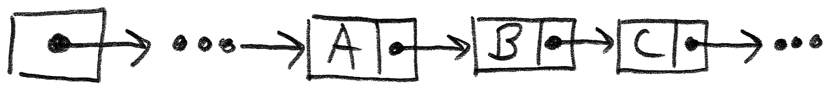
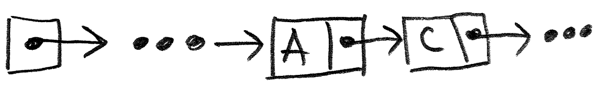


what are the steps?
write pseudo-code on paper.
Does it work for both cases?

Linked Lists

```
INSERT ( data ):
    If data is already stored return failure
    Allocate new Node
    Set new Node data to incoming data
    Set now Node next to current head
    Set head to new Node
    Return success.
```

Example of DELETE(B):

Case	Before	After
First Node		
Last Node		
Middle Node		

What are the steps?

Write pseudo-code on paper.

Does it work for all cases?

What if B isn't stored?

Linked Lists

```
DELETE(key):
    Set current Node to head node
    set previous Node to null pointer
    until the end is found:
        if current Node's data matchos key:
            if no previous Node:
                set head Node to current Node's next

            else:
                set previous Node's next to
                    current Node's next.

            deallocate current Node.

            return success
        set previous to current
        set current to current's next
    return failure
```