CS 2113
Software Engineering

Do this now!!!

From C to Java

```
git clone https://github.com/cs2113f18/c-to-java.git
                    cd c-to-java
                   ./install_java
```

Previously...

- We finished up C
  - There is plenty more to learn, but you've had a taste
- You are completing Module 3
  - linked lists and more complex data structures

This Time…

- A bit more C
- More on Linked Lists
- Algorithmic thinking, APIs
- Going from C to Java

Final notes on C

- The benefits of C:
  - Low level coding
  - Direct access to memory
  - Ubiquitous
  - Low overhead

- The dangers of C:
  - Direct access to memory
  - Minimal type checking
  - No support for objects
  - No variable initialization

Final notes on C

- Remember the memory model
  - This is not C specific
  - But other languages hide the details
- Most C bugs are related to how you access memory
  - If in doubt... draw it out!
- How to learn a new language:
  - Small steps!
  - Write code, compile, test, repeat
  - Look at library reference examples

source: Head First C

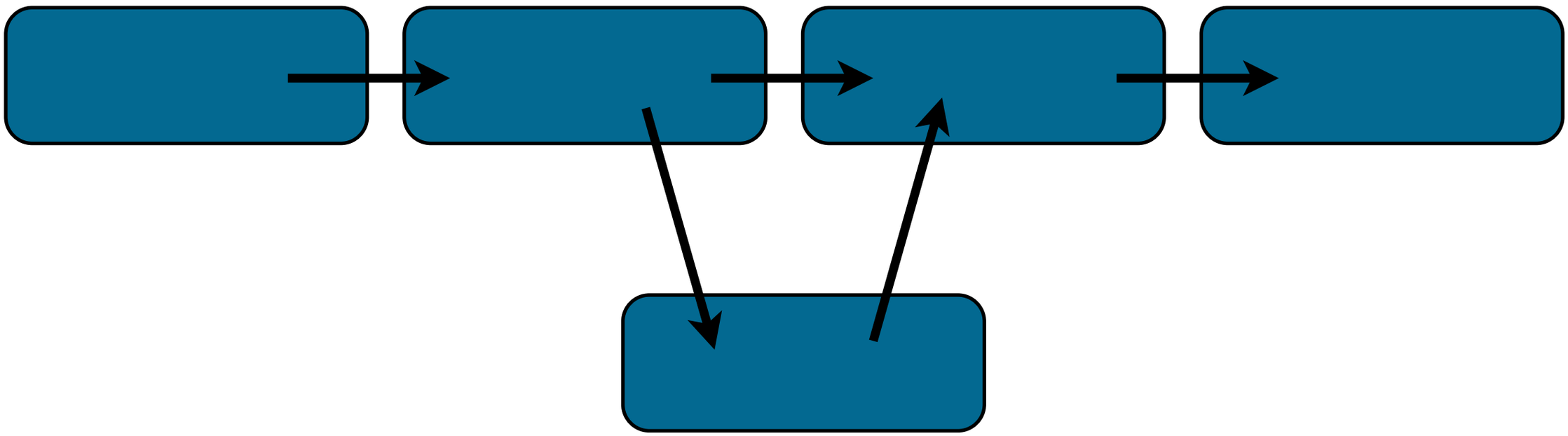
Plan big, code small

- Plan your overall approach
  - Write pseudo code for your algorithm
  - Figure out what data, functions, objects you will need
  - Break the problem into small pieces

- Write code piece by piece
  - Never try to write your whole program at once
  - Write a small piece and test it out
  - Move to the next step when you know one piece works

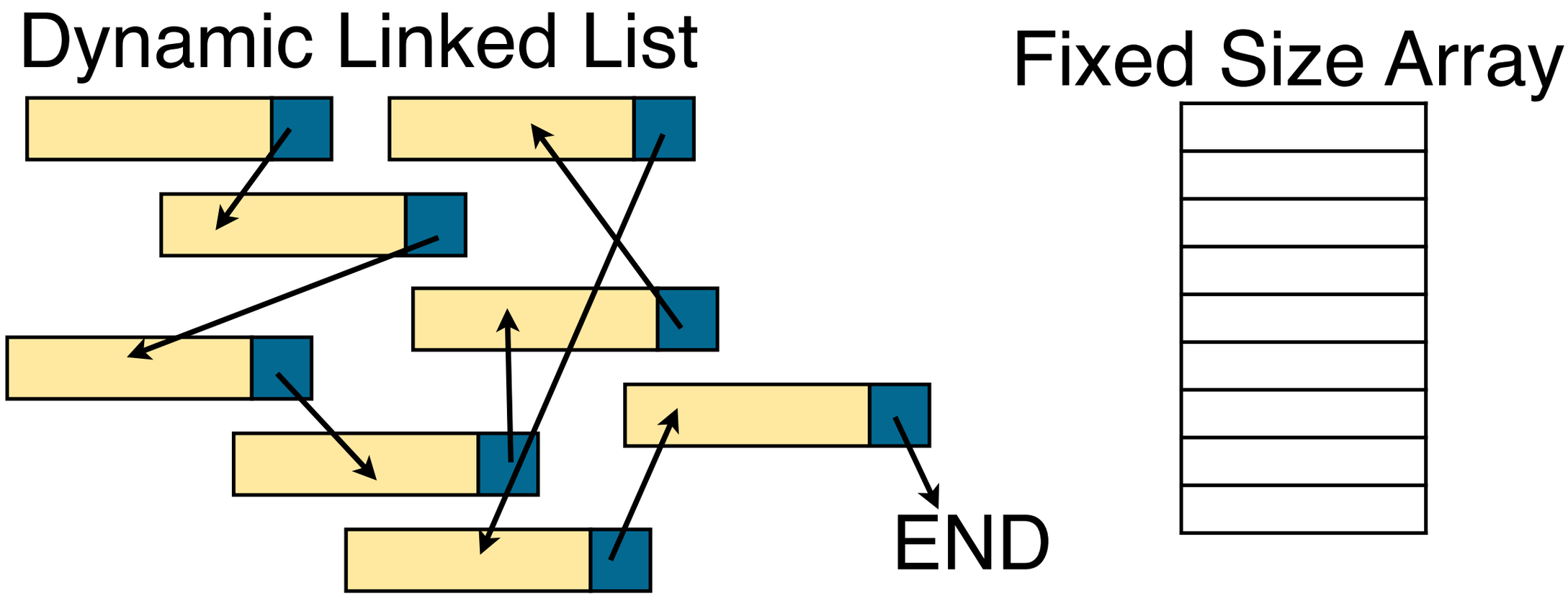
The Linked List

- What is a linked list?
- What can it hold?
- How does it compare to...
  - An array from the stack? `int days[365];`
  - or the heap? `int *days=malloc(365*sizeof(int));`



The Linked List

- Strength: Very flexible
- Weakness: Slow access time



What functions do we need?

- A linked list should be able to...
  - create
  - search
  - delete
  - insert - in middle, at end, etc
  - copy the full list
  - check if empty
  - how many elements?
  - retrieve data (don't want the list to be specific to the data type)

What functions do we need?

- A linked list should be able to...

Application Program Interface

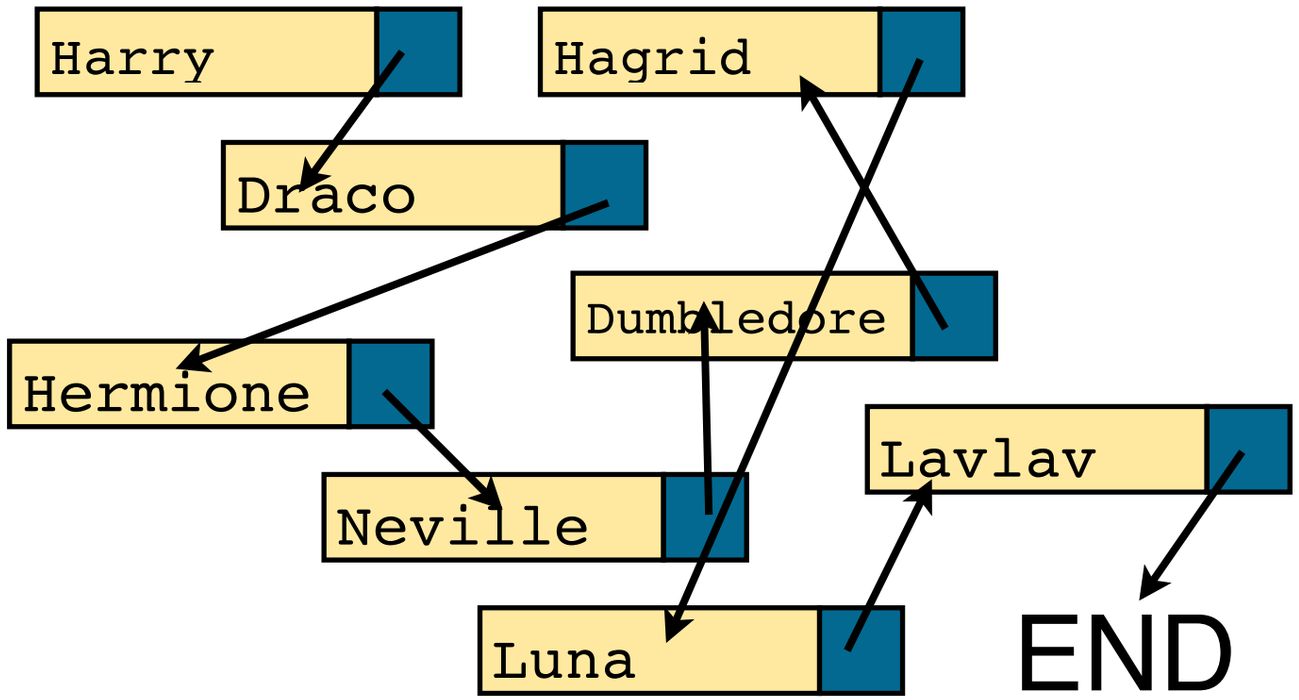
- We just did **software engineering**!
  - SE is about a lot more than writing code and knowing syntax
- An API describes an interface
  - What functionality is exposed? What data is available?
- Is our Linked List interface C-specific?

What data do we need?

- How should we represent the list?

Dynamic Linked List

- Linked List
  - pointer to first Node
- Node
  - String name
  - pointer to next Node



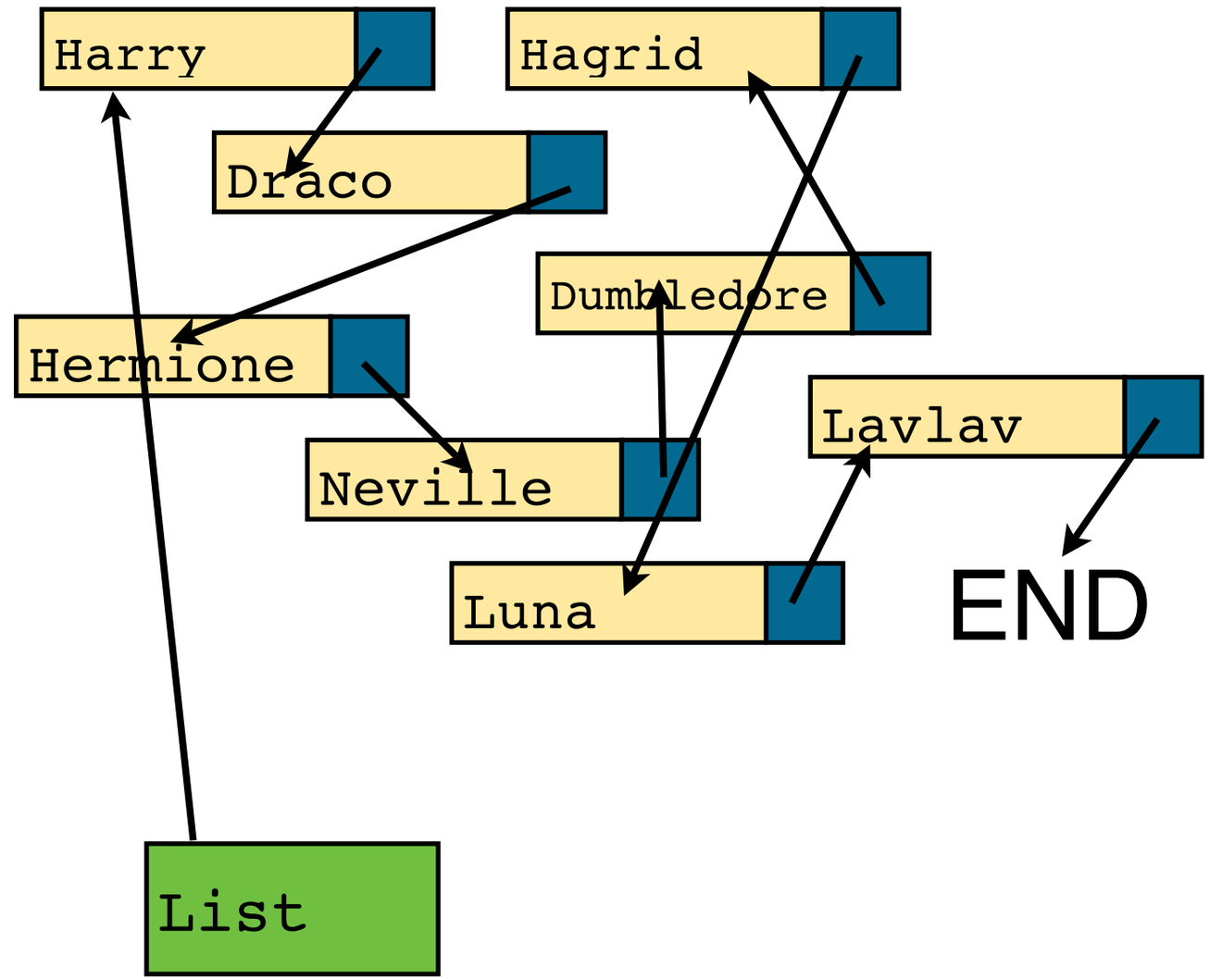
What data do we need?

- How should we represent the list?

Dynamic Linked List

- Node data type
  - Name
  - House
  - Wand type
  - Next node

- List data type
  - First Node in list



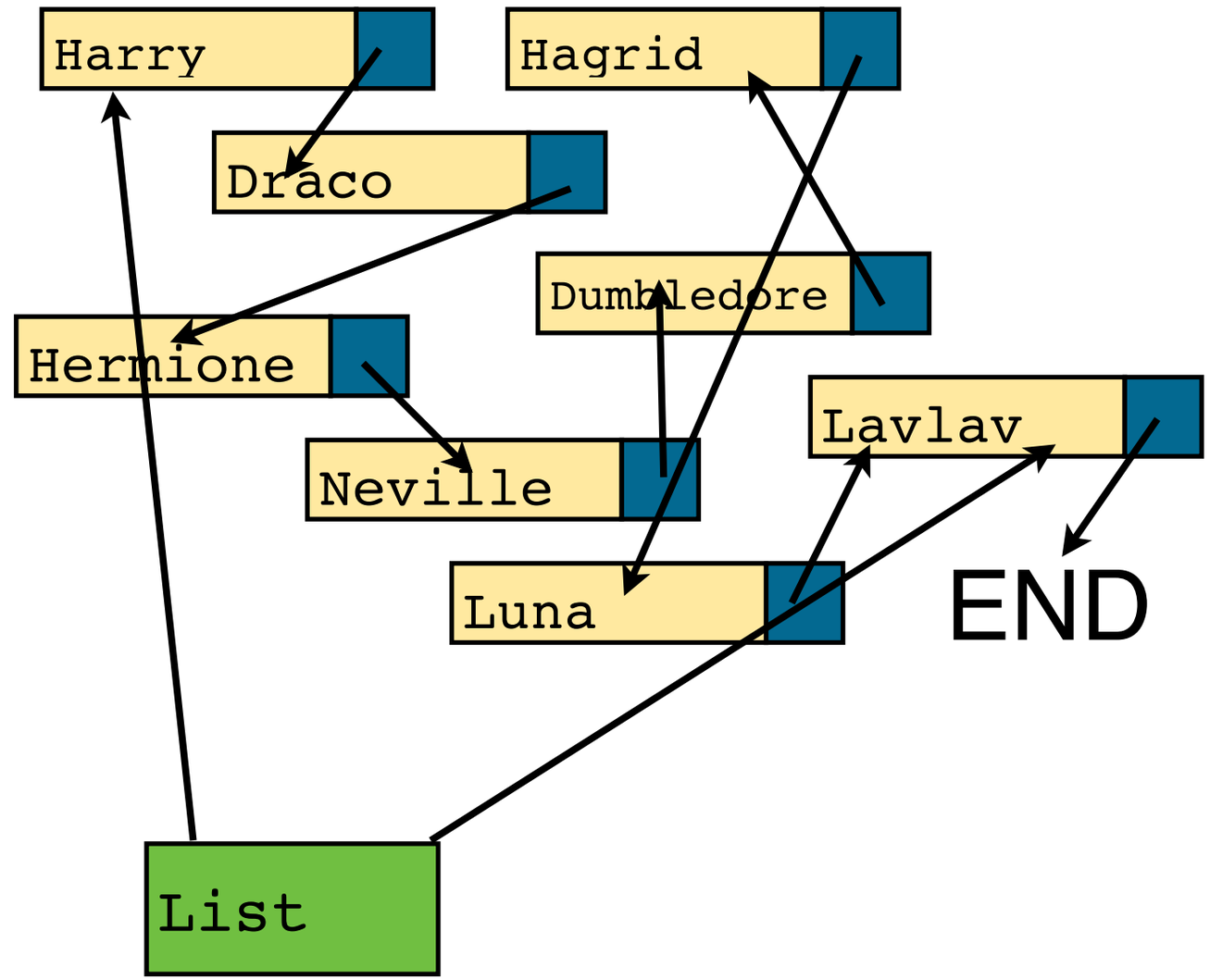
What data do we need?

- How should we represent the list?

Dynamic Linked List

- Node data type
  - Name
  - House
  - Wand type
  - Next node

- List data type
  - First Node in list
  - Last Node in list
  - Count of nodes, etc



LL: Functions + Data

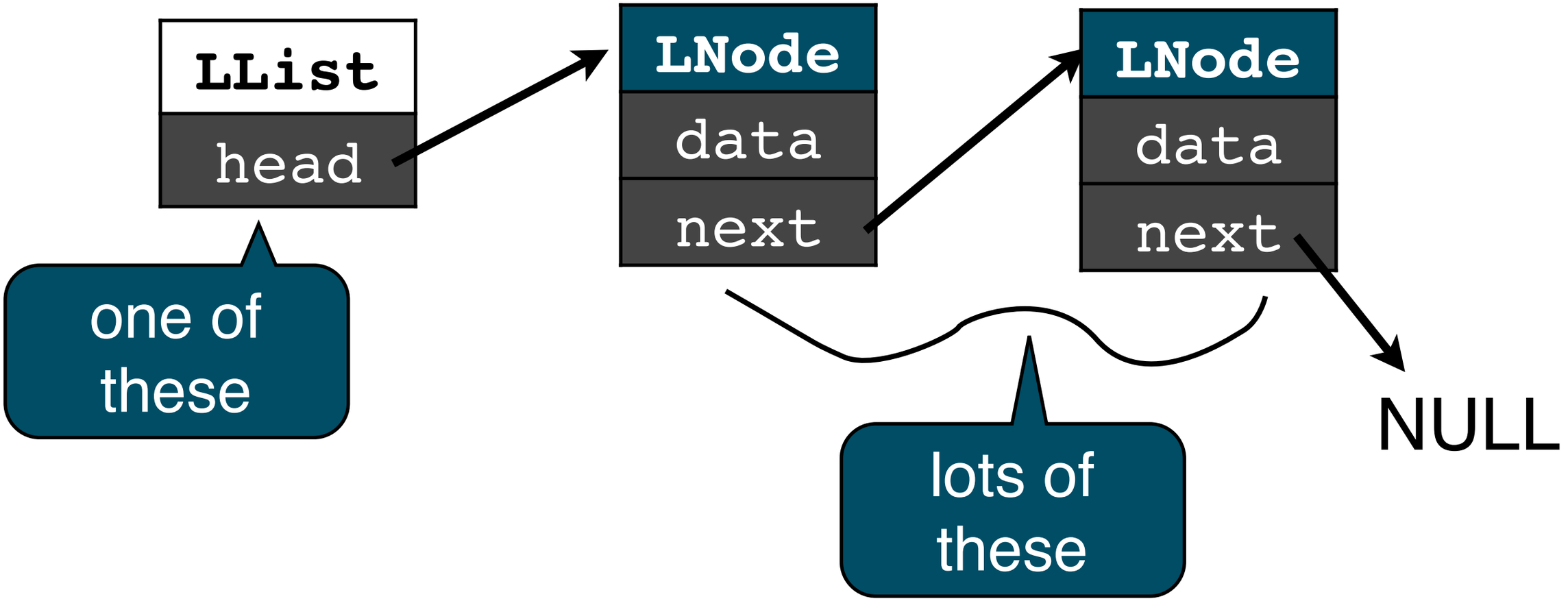
- Add a new element at the end
- Add a new element in sorted order
- Delete a specific element
- Delete all elements
- Print all the elements
- Return the length of the list
- Create a new empty list

- Node data type
  - Name
  - House
  - Wand type
  - Next node

- List data type
  - First Node in list

A Linked List in C

- We will use two types of structs
  - **`LList`**: represents the list as a whole, used by application
  - **`LNode`**: used for each entry in the list, stores actual data
- This gives a nicer API than requiring programmer to understand internals of LNodes



A Note on NULL

- NULL is a reserved keyword in C
  - Often used as a "**sentinel**" to tell whether a pointer has been initialized

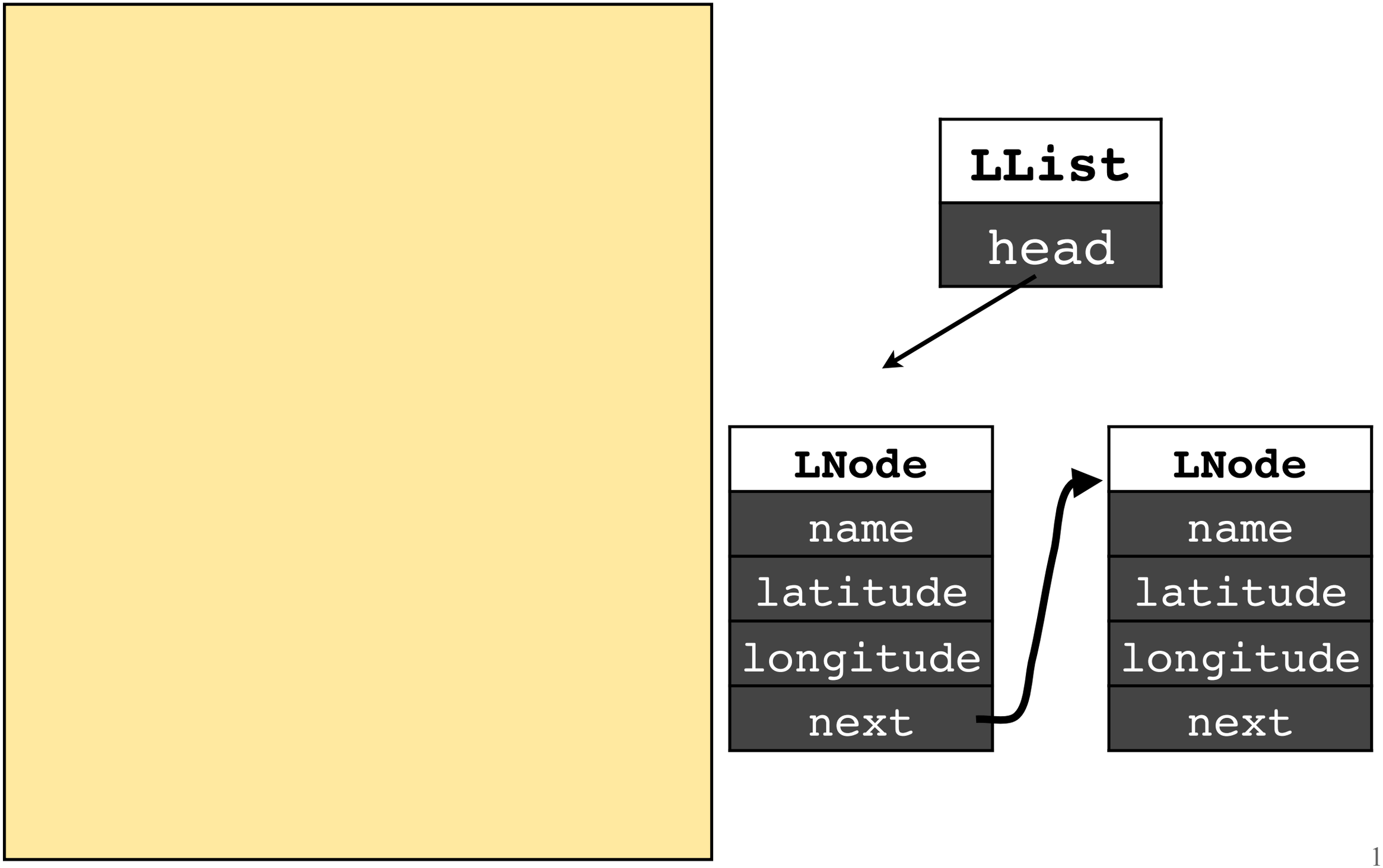
- Are undefined variables automatically set to NULL in C?
  - No!

- We will have to carefully set pointers to NULL by ourselves!

- Secret: NULL is actually just the number 0!

Coding a Linked List

- What is a linked list made of?

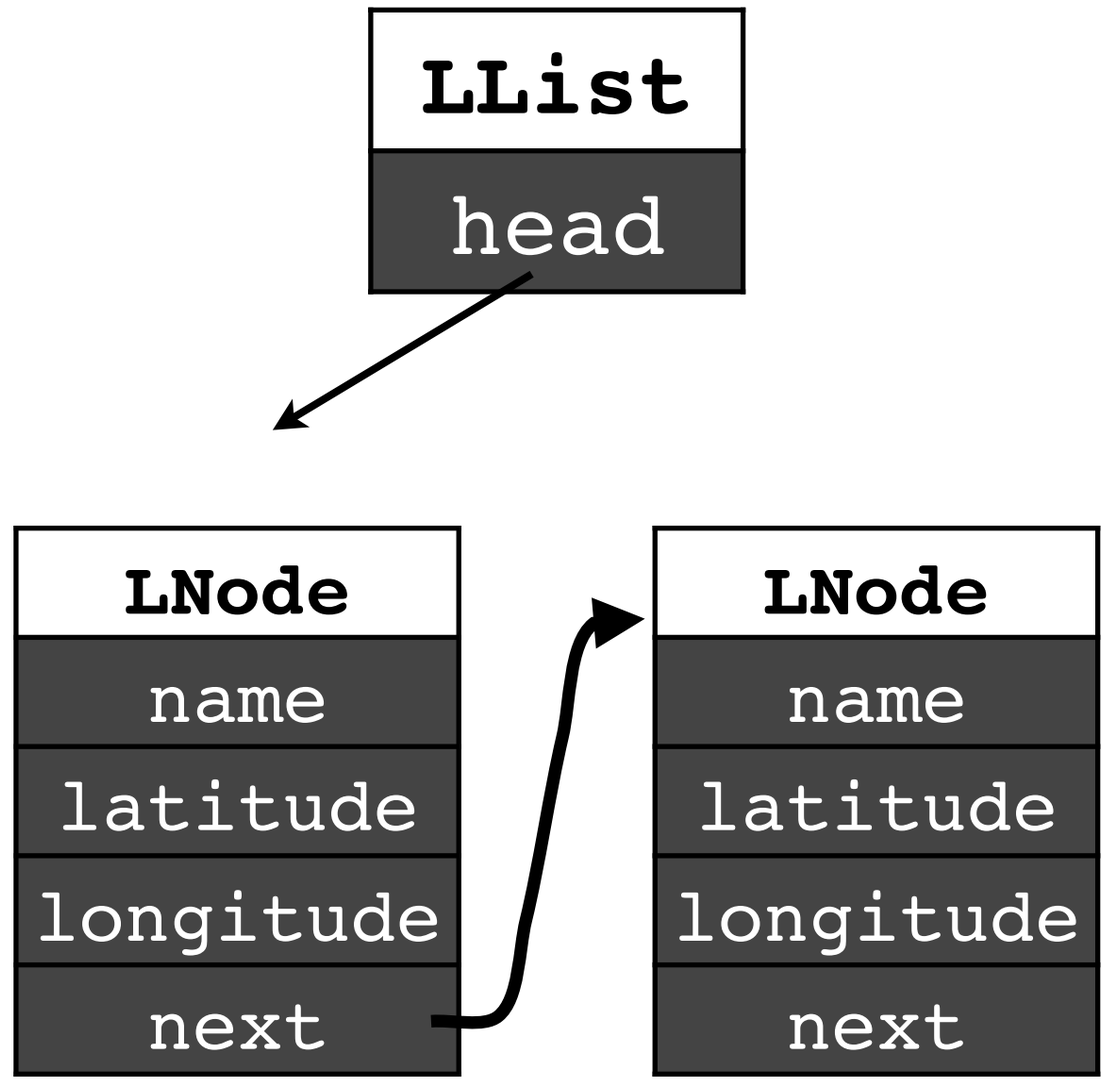


Coding a Linked List

- What is a linked list made of?

```
struct LNode {
   char name
   double latitude
   longitude
   LNode *next
};

struct LList {
   LNode *head;
};
```



Linked Lists and Memory

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

```
struct LList {
  LNode* head;
};
```

```
int main() {
 struct LList* list;
 struct LNode *a, *b;
 list = NULL;
 a = NULL; b = NULL; c = NULL;

}
```

Stack

Address	*Name*	*Contents*
10000		
10004		
10008		
10012		
10016		

Heap

Address	*Alloc?*	*Contents*
50000		
49996		
49992		
49988		
49984		
49980		
49976		
49972		
49968		

Assume ints and pointers take 4 bytes.

Linked Lists and Memory

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

```
struct LList {
  LNode* head;
};
```

```
int main() {
 struct LList* list;
 struct LNode *a, *b;
 list = NULL;
 a = NULL; b = NULL; c = NULL;

list = (struct LList*) malloc(sizeof(LList));
a = (struct LNode*) malloc(sizeof(struct LNode));
b = (struct LNode*) malloc(sizeof(struct LNode));
c = (struct LNode*) malloc(sizeof(struct LNode));
a->data = 45;
b->data = 89;
c->data = 52;
list->head = a;
a->next = b;
b->next = c;
c->next = NULL;
```

Stack

Address	*Name*	*Contents*
10000	list	50000
10004	a	49996
10008	b	49988
10012	c	49980
10016		

Heap

Address	*Used?*	*Contents*
50000	Y	head: 49996
49996	Y	a->data: 45
49992	Y	a->next: 49988
49988	Y	b->data: 89
49984	Y	b->next: 49980
49980	Y	c->data: 52
49976	Y	c->next: NULL
49972	N	
49968	N	

Assume ints and pointers take 4 bytes.

Linked Lists and Memory

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

```
struct LList {
  LNode* head;
};
```

```
int main() {
 struct LList* list;
 struct LNode *a, *b, *c;
 list = NULL;
 a = NULL; b = NULL; c = NULL;
 list = malloc(sizeof(struct LList));
 a = malloc(sizeof(struct LNode));
 b = malloc(sizeof(struct LNode));
 c = malloc(sizeof(struct LNode));
 list->head = a;
 a->data = 45;
 a->next = b;
 b->data = 89;
 b->next = c;
 c->data = 52;
 c->next = NULL;
}
```

Remember, really memory just holds data, not names!

Stack

Address	*Name*	*Contents*
10000	list	50000
10004	a	49996
10008	b	49988
10012	c	49980
10016		

Heap

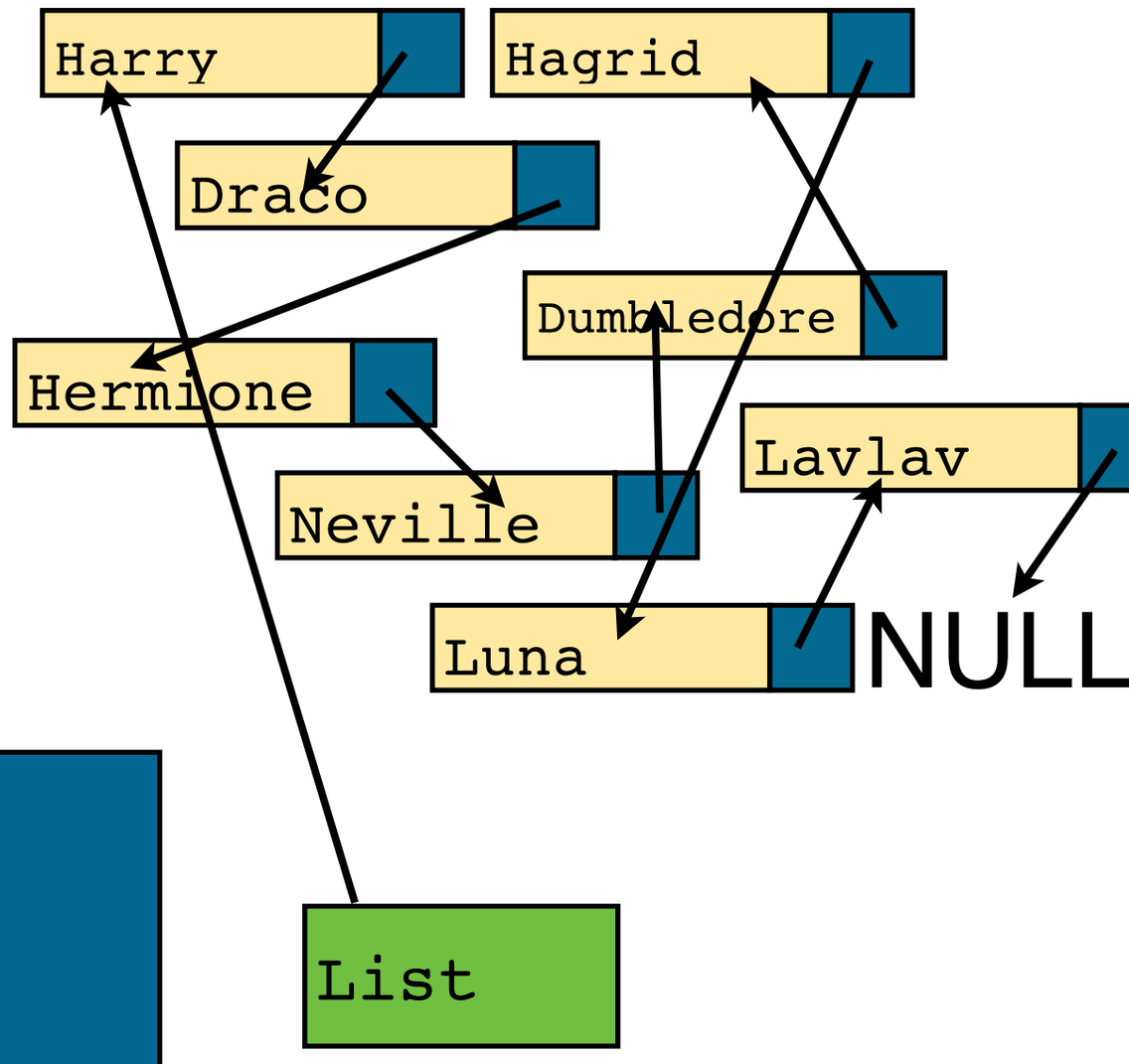
Address	*Alloc?*	*Contents*
50000	Y	49996
49996	Y	45
49992	Y	49988
49988	Y	89
49984	Y	49980
49980	Y	52
49976	Y	0
49972	N	
49968	N	

Assume ints and pointers take 4 bytes.

Algorithm to print a LList

- What steps do we need to take?
  - Don't worry about C syntax

```
// in class solution
input: list we want to print
return if the list is empty
go to first node and print it
while there is a next node
   go to the next node
  print that node
```



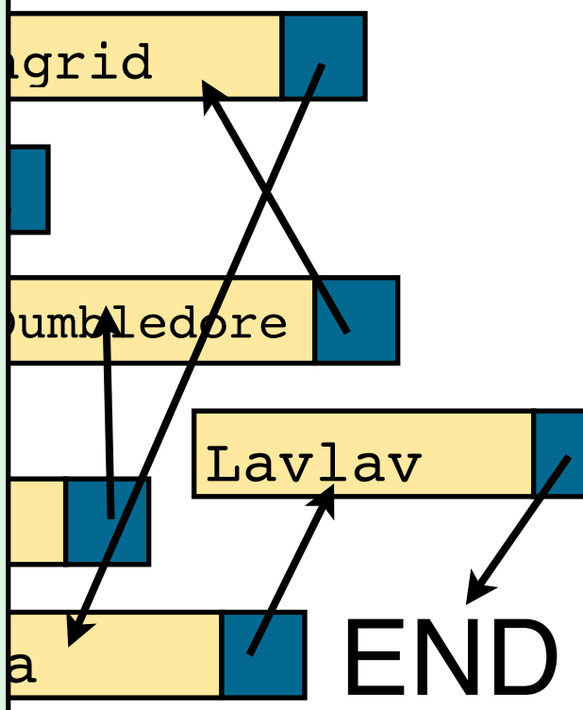

```
Edge cases:
 - last node (include and stop)
 - empty list
```

Algorithm to print a LList

- What steps do we need to take?
  - Don't worry about C syntax

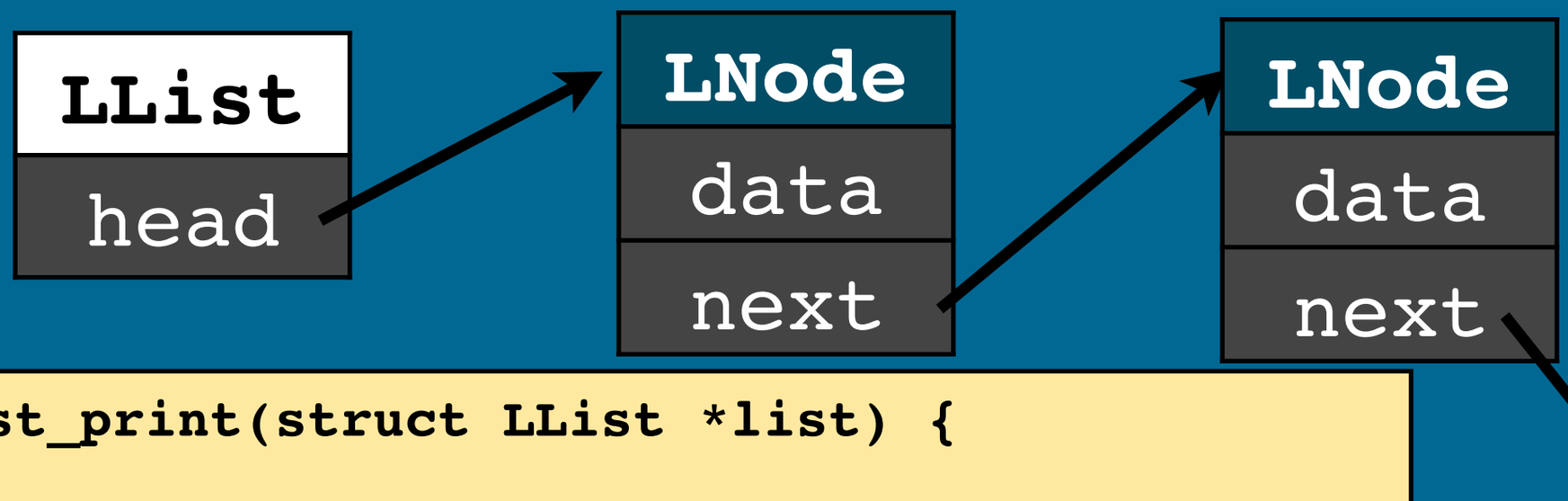
Point at the first node in the list

Start loop...
Print out the data for the current node
If the next node in the list is empty, exit
Else, point at the next node in the list
Go back to start of loop



Edge cases:
Uninitialized List
Empty list
End of list

Printing out a L.L.

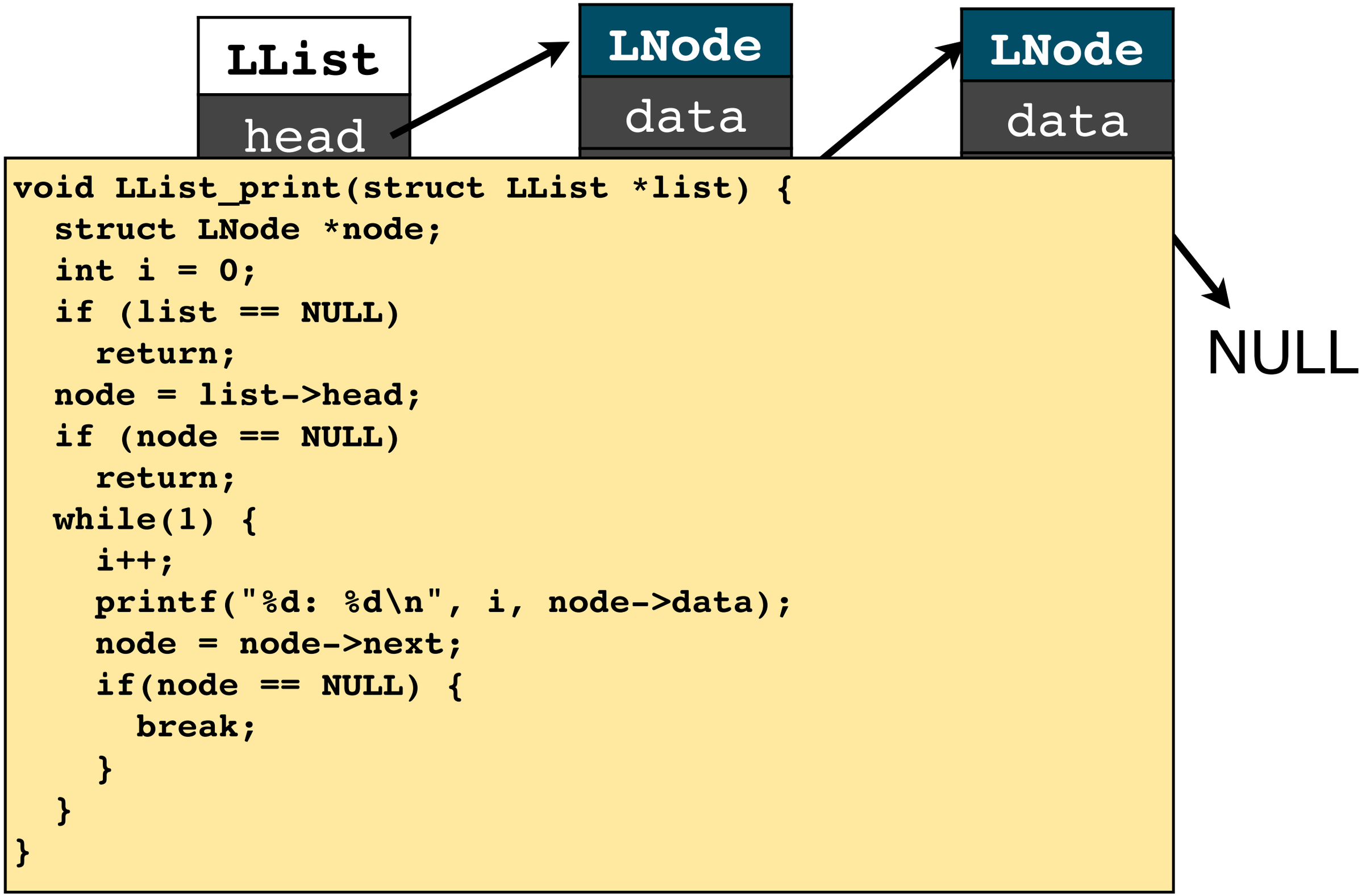



```
void LList_print(struct LList *list) {
{
    /* Print a list from head to tail. */
    if(list == NULL || list->head == NULL) return;
    struct LNode *p = list->head;
    print(p);
    while(p->next != NULL) {
        p = p->next;
        print(p);
    }

}
```

input: list we want to print
return if the list is empty
go to first node and print it
while there is a next node
* go to the next node*
* print that node*

Printing out a L.L.




```
void LList_print(struct LList *list) {
  struct LNode *node;
  int i = 0;
  if (list == NULL)
    return;
  node = list->head;
  if (node == NULL)
    return;
  while(1) {
    i++;
    printf("%d: %d\n", i, node->data);
    node = node->next;
    if(node == NULL) {
      break;
    }
  }
}
```

Algorithm for append

input: list we want to add to and some new data

if list is empty:
add new element and make it the head of the list
return
go to first node
while there is a next node
go to the next node
(now we are at the last node in the list)
create a new element, and set it as the next node
mark this node as the last in the list

Edge cases:

Algorithm for append

Add node to end of a list:
inputs: List to add to, data to store inside node

allocate memory for new Node
Fill data into newNode
Set newNode->next = NULL
if(head of list is NULL)
  point list->head to newNode
else
  step through the list until we reach the end
  set the last entry's next pointer = newNode

Edge cases:
Uninitialized List, Empty list, End of list
Out of memory for new Entry

C-ish Languages

- C++
  - Enhances C with support for objects and classes
  - Adds the Standard Template Library (STL) for data structures
  - Slightly more flexible language
  - Just as powerful... just as dangerous
- Objective C and Swift
  - Primarily used by Apple
  - Superset of C
  - Adds objects to C in a more confusing way than C++ / Java
  - Extensive library support and custom IDE makes it more bearable
    - So does the potential for earning millions on the App Store!

Moving to Java

- Java Syntax
  - You should already know this...
  - Use book to refresh on basics

- The textbook is "Head First Java" (2005 edition)
  - Readings will be assigned each week
  - **Read them before LAB**
  - **or else...**

Midterm

- Mix of written on paper and coding on computer
  - (If I can get access to a second computer lab)
- List of topics:
  - https://cs2113f18.github.io/midterm.html
- Practice Problems:
  - https://cs2113f18.github.io/c/review.html
- C Reference Sheet:
  - https://cs2113f18.github.io/c/c-reference.pdf