

CS 2113

Software Engineering

Do this now!!!

Lecture 5:
From C to Java

```
git clone https://github.com/cs2113f16/lec-5.git
cd lec-5
./install_java
```

Previously...

- We finished up C
 - There is plenty more to learn, but you've had a taste
- You completed Exercise 4
 - strings, file I/O, and memory allocation
- You got some practice with a debugger
 - This is possibly the most useful skill you can develop
 - The better you get at programming, the more code you will write that doesn't work...

This Time...

- A bit more C
- More advanced data structures
- Algorithmic thinking, APIs
- Going from C to Java

Ex 4

- Computerized Poetry
- Be sure you understand the difference between:

```
char line[200]; // 200 letters  
line[20] refers to a single character
```

```
char *lines[100]; // 100 strings  
lines[20] refers to a string, must call  
malloc before accessing it!
```

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

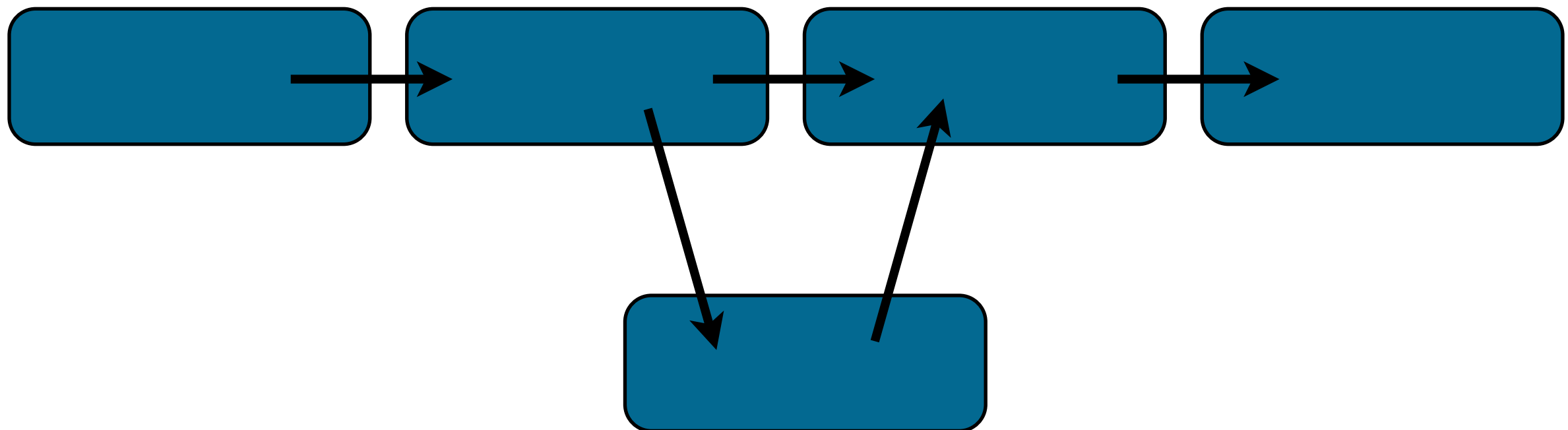


Plan big, code small

- 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));`

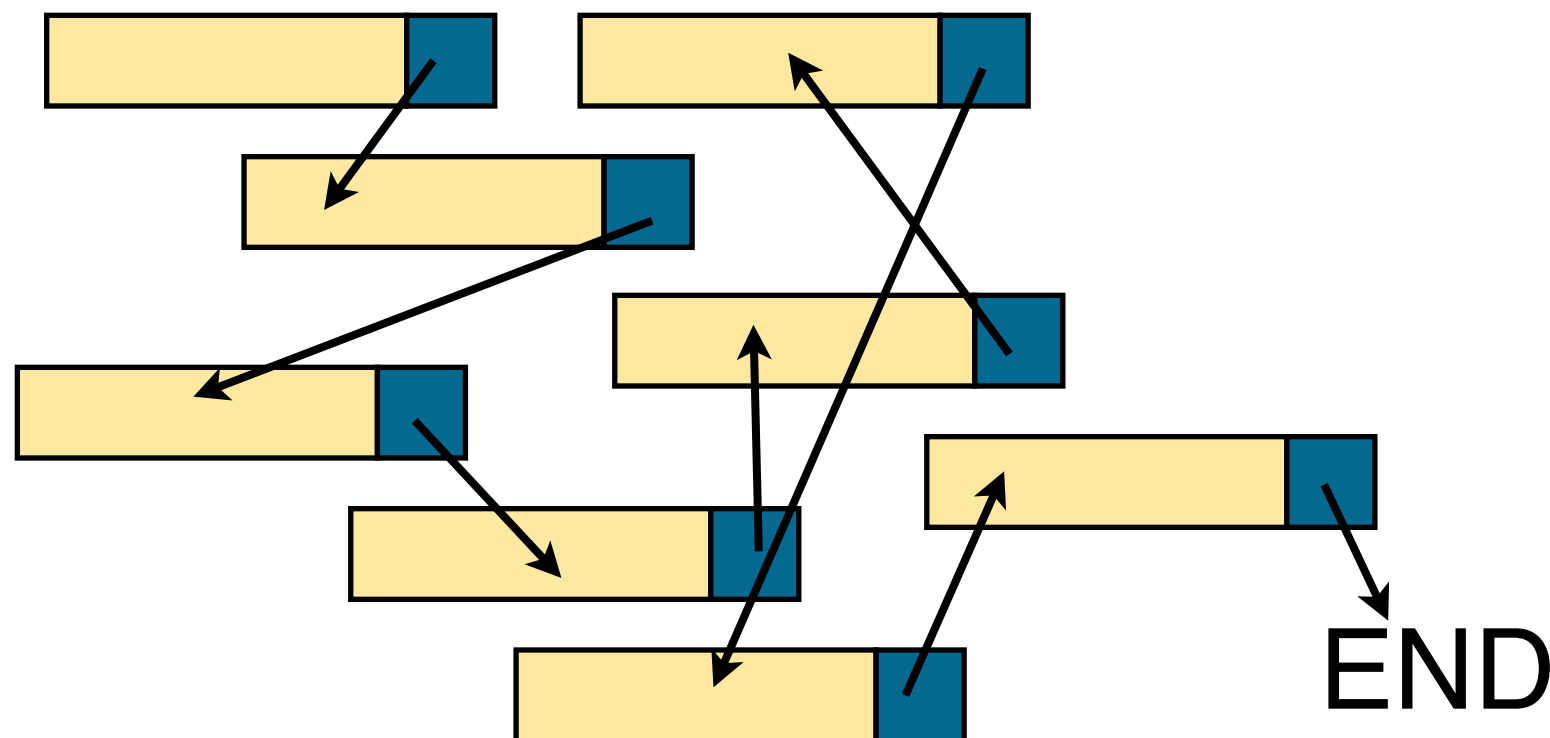


The Linked List

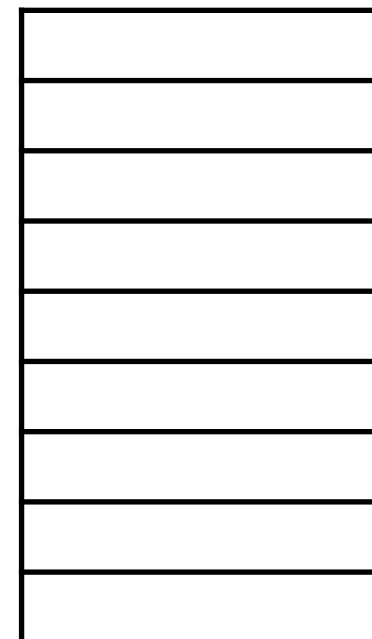
- Strength: Very flexible

- Weakness: Slow access time

Dynamic Linked List



Fixed Size Array



What functions do we need?

- A linked list should be able to...

What functions do we need?

- A linked list should be able to...
 - Add a new element at the end
 - Add a new element at the start
 - Add a new element in sorted order
 - Add an element at a specific location
 - Delete a specific element
 - Delete all elements
 - Delete the last N elements
 - Delete the first N elements
 - Print all the elements
 - Return the length of the list
 - Create a new empty list

Application Program Interface

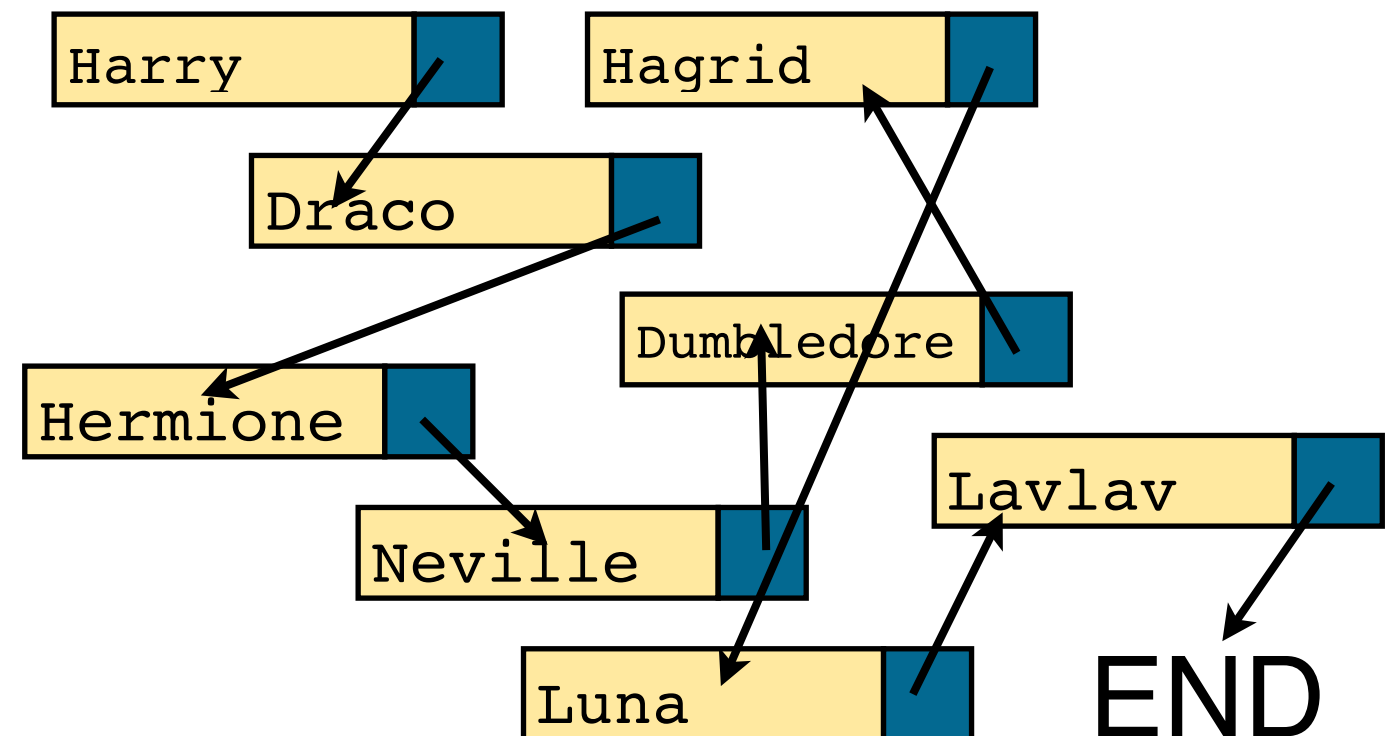
- We just did **software engineering!**
 - SE is about a lot more than writing code and knowing syntax
- An API describes an interface
 - What functionality is exposed? What data is available?
- Is our Linked List interface C-specific?

What data do we need?

- How should we represent the list?

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

Dynamic Linked List



What data do we need?

- How should we represent the list?

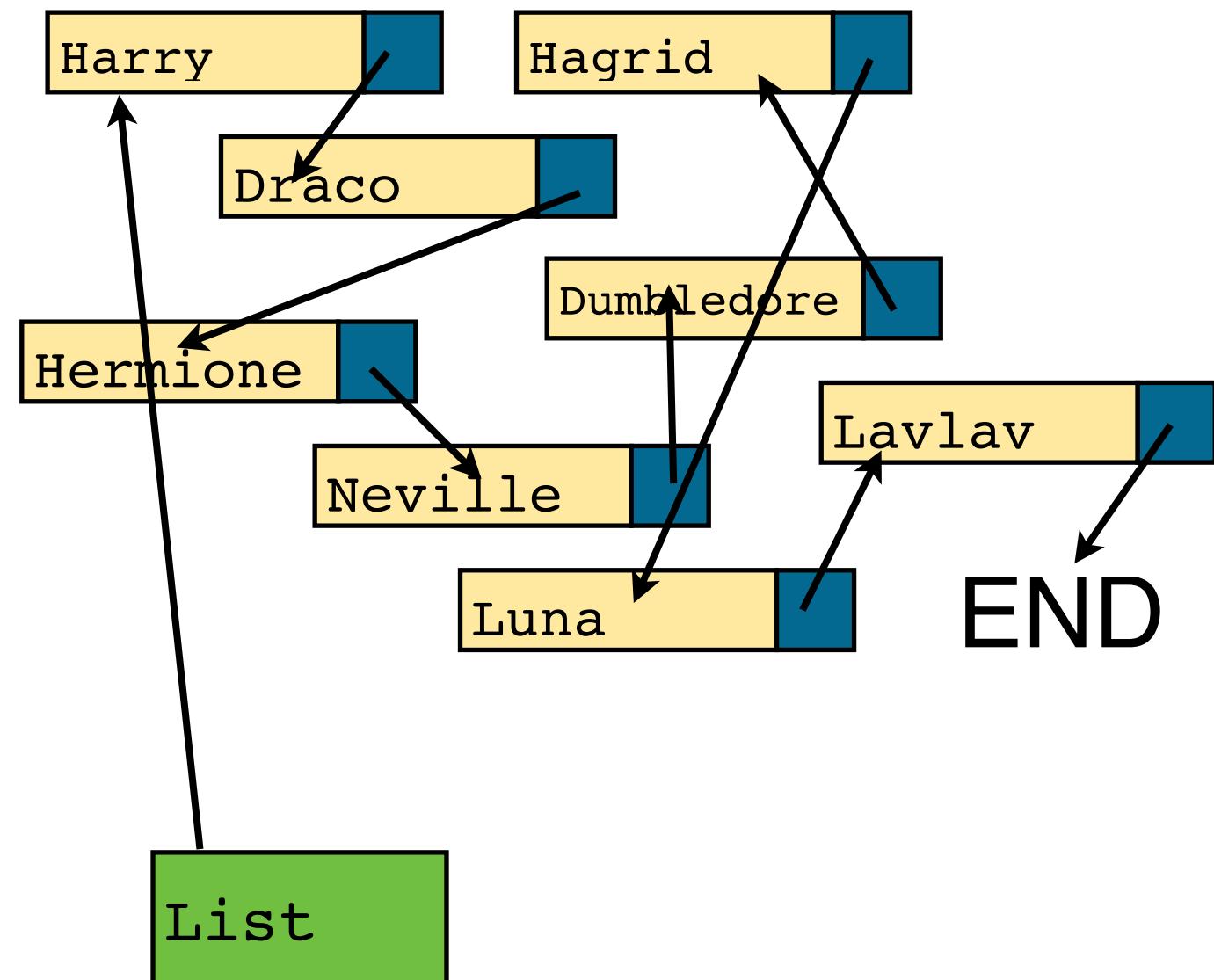
- Node data type

- Name
- House
- Wand type
- Next node

- List data type

- First Node in list

Dynamic Linked List



What data do we need?

- How should we represent the list?

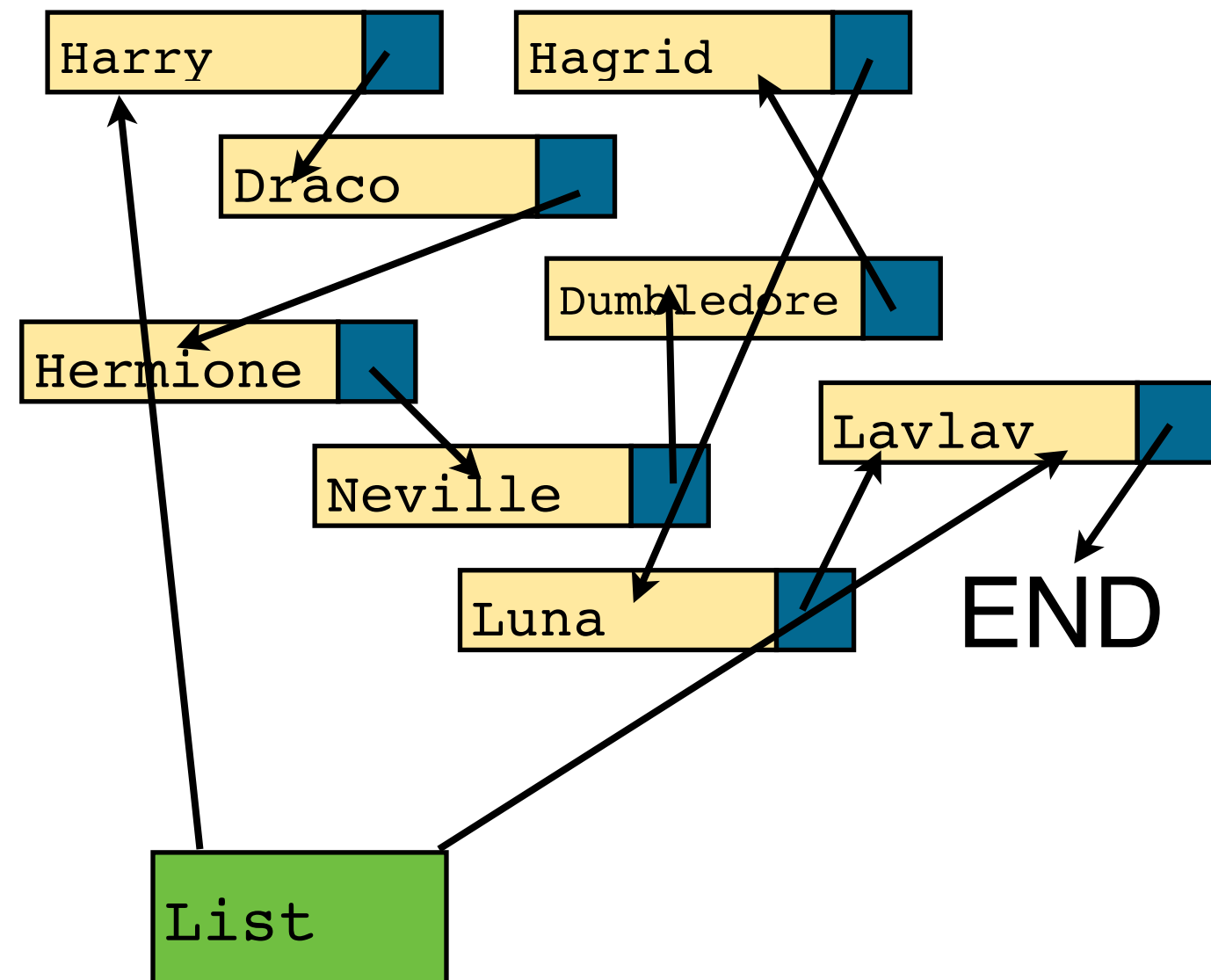
- Node data type

- Name
- House
- Wand type
- Next node

- List data type

- First Node in list
- Last Node in list
- Count of nodes, etc

Dynamic Linked List

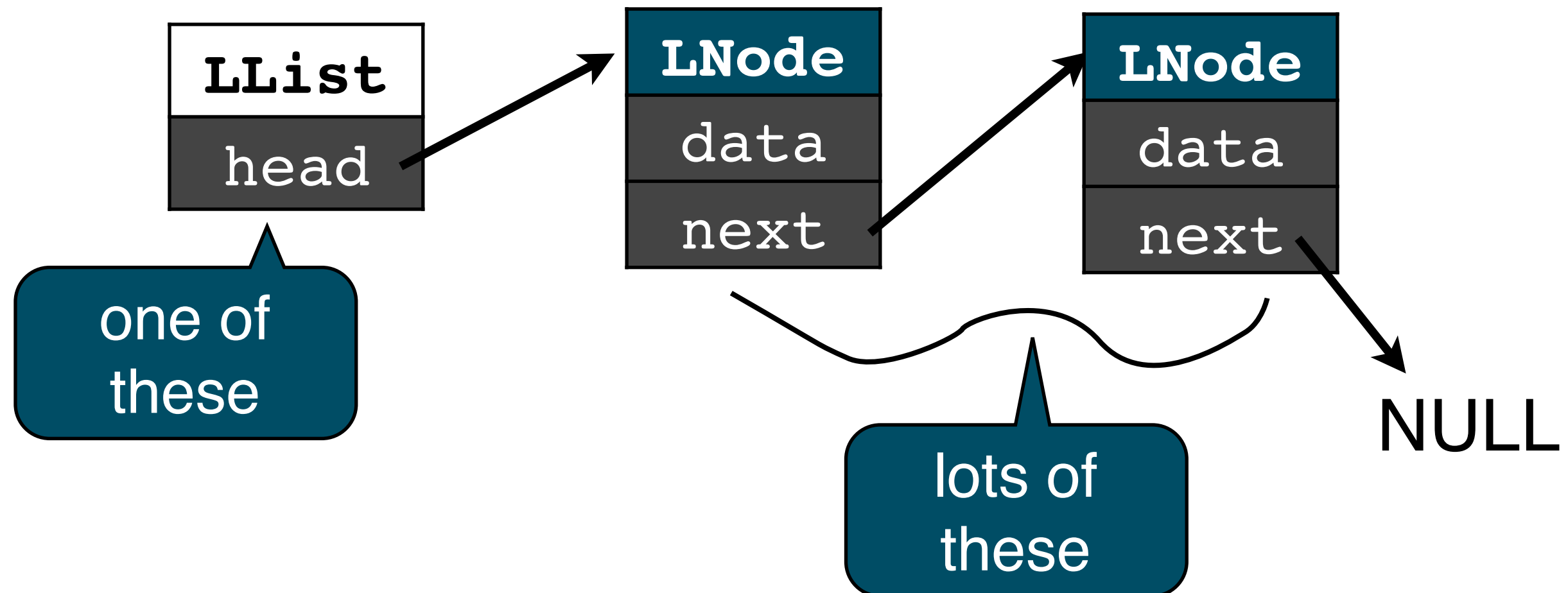


LL: Functions + Data

- 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

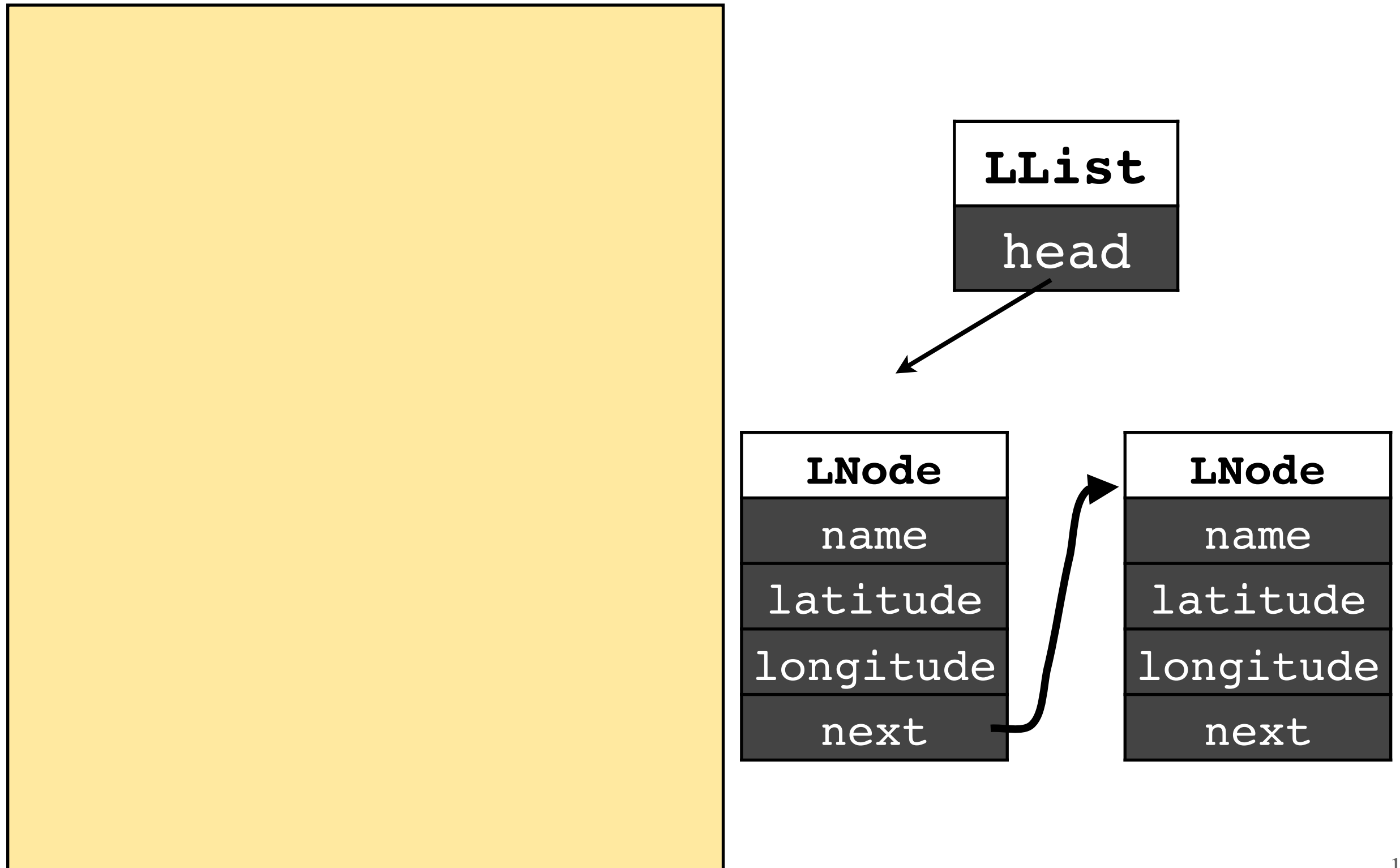


A Note on NULL

- 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

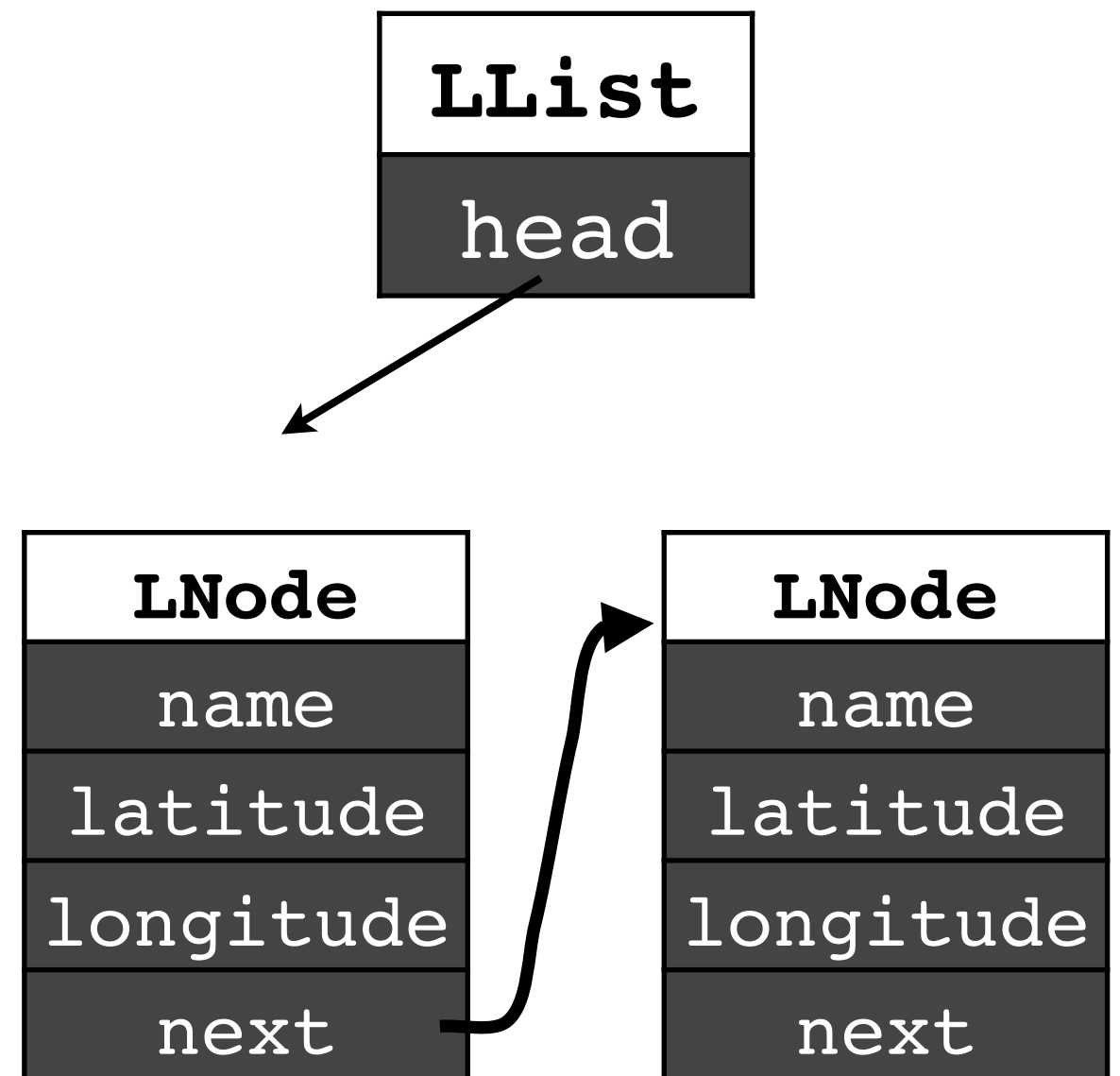
- What is a linked list made of?



Coding a Linked List

- What is a linked list made of?

```
struct LList {  
    struct LNode* head;  
  
};  
  
struct LNode {  
    char name[20];  
    float lat;  
    float lon;  
    struct LNode* next;  
  
};
```



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, *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;  
}
```

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

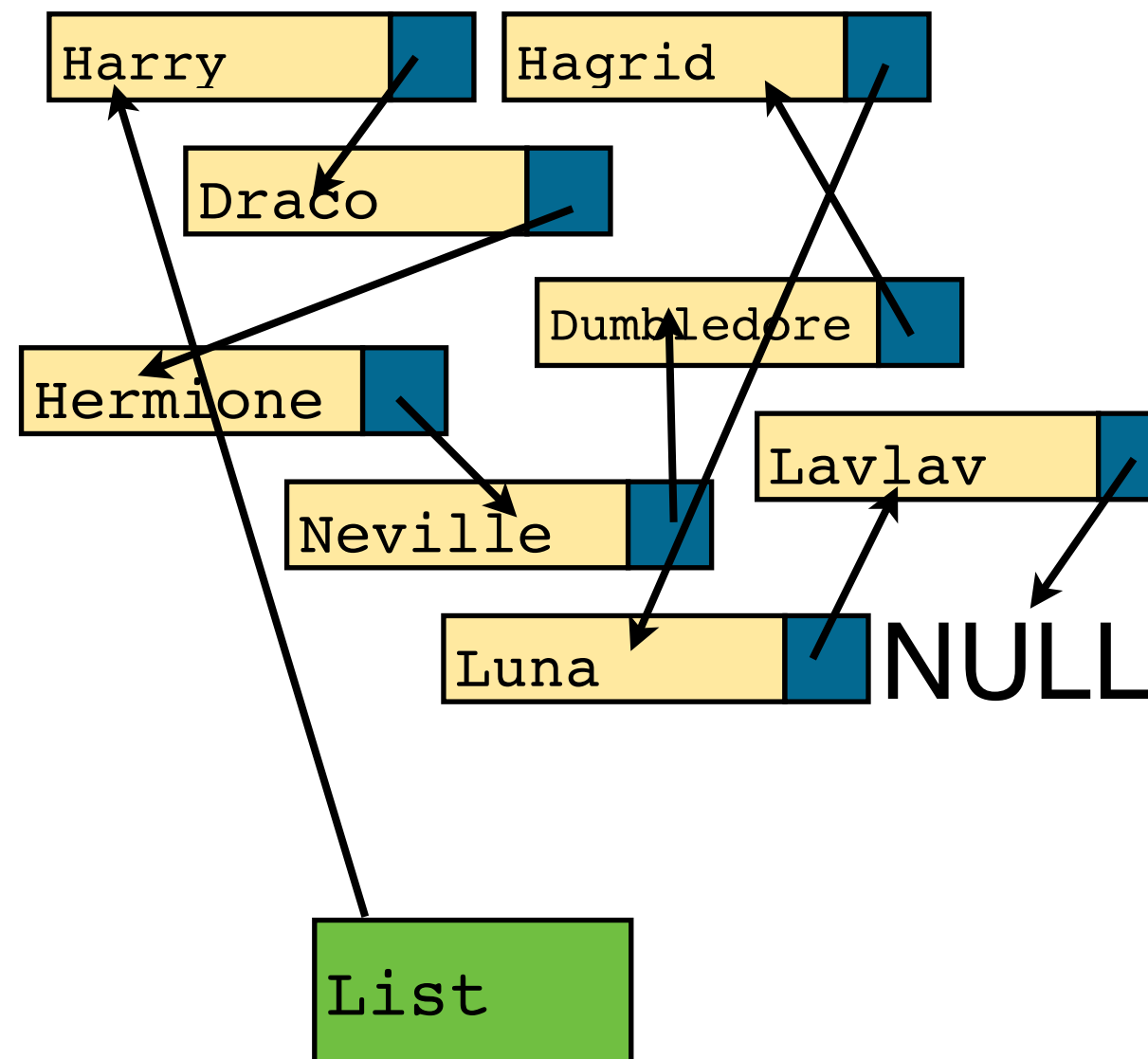
Heap		
Address	Alloc?	Contents
50000	Y	49996
49996	Y	45
49992	Y	49988
49988	Y	89
49984	Y	49980
49980	Y	52
49976	Y	0
49972	N	
49968	N	



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



Algorithm to print a LLlist

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

Point at the first node in the list

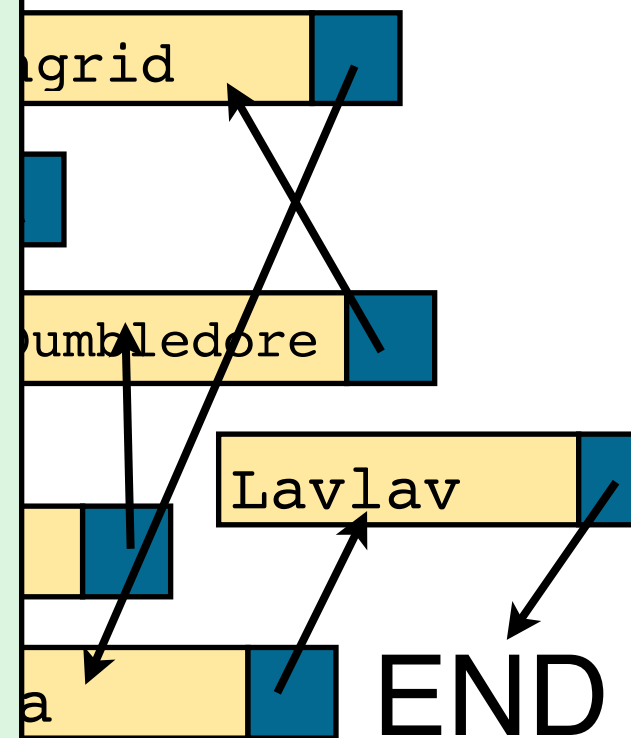
Start loop...

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



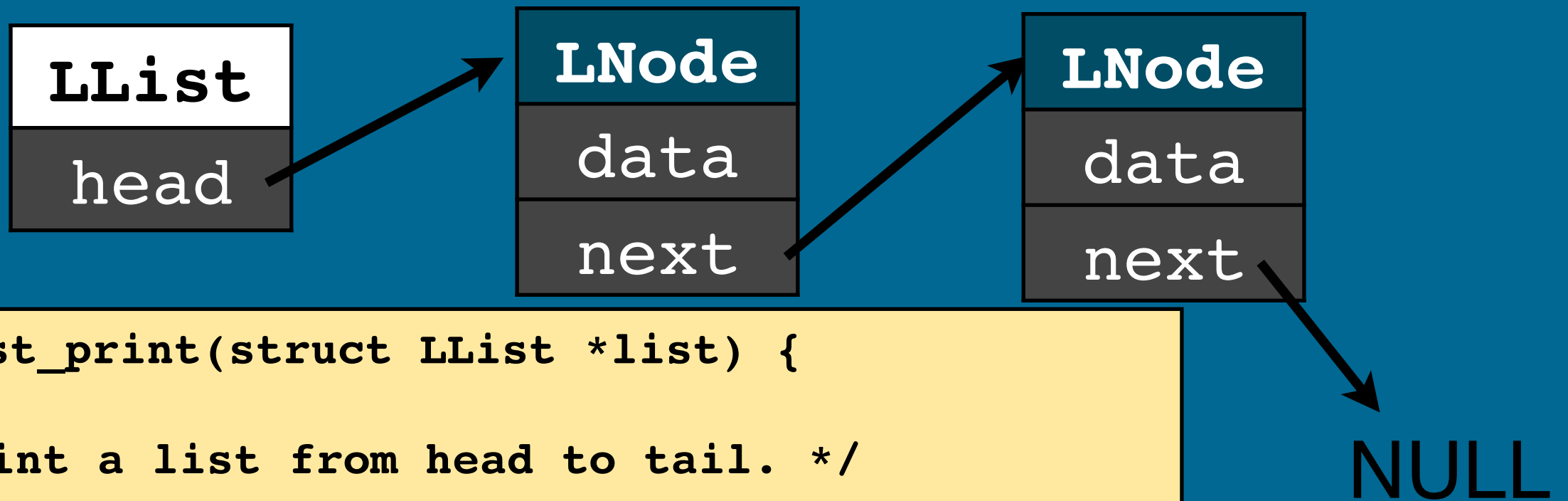
Edge cases:

Uninitialized List

Empty list

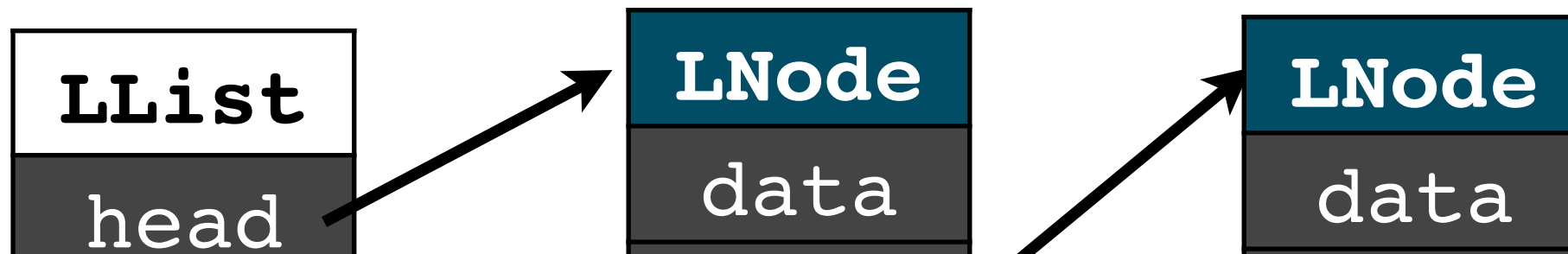
End of list

Printing out a L.L.



```
void LList_print(struct LList *list) {  
    /* Print a list from head to tail. */  
  
    // PUT YOUR CODE IN lec-5/llist.c  
  
}
```

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;
        }
    }
}
```

NULL

Algorithm for append

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 new Node

Set new Node->next = NULL

if(head of list is NULL)

point list->head to new Node

else

step through the list until we reach the end

set the last entry's next pointer = new Node

Edge cases:

Uninitialized List, Empty list, End of list

Out of memory for new Entry

Proj. 1: Sorted Linked List

- Make a linked list for storing GPS locations:
- `addSortedNode(...)`
 - Add a new point in order sorted by its longitude (lower to higher)
- `remNode(int index)`
 - Remove the point at the specified index
- `delLList(struct LList* list)`
 - Free all of the memory currently used by list

Implement in both C and Java!

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...**

Java

- What is it?
 - Object oriented programming language
 - A compiler and run time environment
- Java Virtual Machine (JVM)
 - Interprets your Java code and translates it for your OS & HW
 - Compile your code once, run it anywhere

Why Java?

"A simple, object-oriented, distributed, interpreted, robust, secure, architecture-neutral, portable, high-performance, multithreaded, dynamic language."

Why Not Java?

- Java takes the "lowest common denominator" approach to supporting OS features
 - JVM must be cross platform, so cannot exploit specifics
- For efficient, high-performance scientific code, it's better to use C/C++
 - Memory management in Java is the main slowdown
- For low-level system stuff, you have to use C/C++
 - The Windows or Linux OS can't be run inside a JVM...

The Real Reason...

- Java has become hugely popular:

Language encourages good programming practices

- Objects, type checking, automated memory management, and run-time system
 - Prevent bugs or make finding and eliminating them easier
- It will be easier for you to write better code, improving productivity
- But, of course, it's not perfect

The Simplest Java

- Sick of "Hello World" yet?

```
public class HelloWorld {  
  
    public static void main (String[] argv)  
    {  
        String s = "Hello World!";  
        System.out.println(s);  
    }  
  
}
```

- Must be put in a file called "HelloWorld.java"
 - Unlike C, file name is important and must match class name
- Program start point: `public static void main(...)`

Primitive Data Types

- Java has the same basic data types as C
 - `short`, `int`, `long`, `float`, `double`, `char`
 - No unsigned types
- Plus two more
 - `byte` (8 byte number) and `boolean` (T or F)
- All data types (except Boolean) have a precisely defined size on all platforms
 - Different from C where size depends on architecture: 16 vs 32 vs 64bit ints
- All variables are initialized to zero, false, or null

Java Primitives

FIGURE 3-1 Primitive types in Java

Type	Domain	Common operations
byte	8-bit integers in the range -128 to 127	<i>The arithmetic operators:</i> + add * multiply - subtract / divide % remainder <i>The relational operators:</i> == equal != not equal < less than <= less or equal > greater than >= greater or equal
short	16-bit integers in the range -32768 to 32767	
int	32-bit integers in the range -2147483648 to 2147483647	
long	64-bit integers in the range -9223372036854775808 to 9223372036854775807	
float	32-bit floating-point numbers in the range $\pm 1.4 \times 10^{-45}$ to $\pm 3.4028235 \times 10^{38}$	<i>The arithmetic operators except %</i> <i>The relational operators</i>
double	64-bit floating-point numbers in the range $\pm 4.39 \times 10^{-322}$ to $\pm 1.7976931348623157 \times 10^{308}$	
boolean	the values true and false	<i>The logical operators:</i> && and or ! not
char	16-bit characters encoded using Unicode	<i>The relational operators</i>

Source: Art & Science of Java <http://people.reed.edu/~jerry/121/materials/artsciencejava.pdf>

Compiling and Running

- **javac** = compiler

```
$ javac HelloWorld.java          # prints nothing on success
$ ls
HelloWorld.class      HelloWorld.java
```

- Differences from C:
 - Standard names---always becomes `<ClassName>.class`
 - Each class gets its own compiled file (not just `a.out`)
 - `.class` file is java bytecode---must be interpreted by the JVM
 - Not platform specific assembly instructions
- **java** = run time

```
$ java HelloWorld          # class name to run
Hello World!
```

Pro Tip

- In Unix/Linux if you want to match several file names you can use the * symbol

```
$ javac *.java  
$ ls  
// Every .java file will now be compiled into .class  
// or you will see lots of errors
```

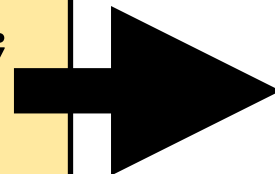

Linked List and Java

- How do we transform C to J

```
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;  
}
```



no structs!
Define a class for each data type

no malloc!
instantiate object with "new"
define constructor to initialize objects

no pointers!
don't need to initialize to null

use . instead of ->

Objects and Classes in Java

- In Java:

- A class is type of object
- All objects have a class
- Primitives (int, float, etc) and functions are not objects


```
public class Hello {  
    public static void main () {  
        System.out.println("hi.");  
    }  
}
```

- Classes contain data and functions

- An object instantiates that data

- Class Constructor

- Used to create new objects
- Initialize variables



```
public class Car {  
    public int x;  
    public int y;  
  
    public Car(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
    public void draw() {  
        // make a picture  
    }  
}
```

Working with objects

```
public class Car {  
    public int x;  
    public int y;  
  
    public Car(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
    public void draw() {  
        // make a picture  
    }  
    public static void main () {  
        Car mercedes;  
  
        mercedes.x = 34;  
        mercedes.y = 11;  
        mercedes.draw();  
    }  
}
```

- Does this code work?

Working with objects

```
public class Car {  
    public int x;  
    public int y;  
  
    public Car(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
    public void draw() {  
        // make a picture  
    }  
    public static void main () {  
        Car mercedes;  
  
        mercedes.x = 34;  
        mercedes.y = 11;  
        mercedes.draw();  
    }  
}
```

- Does this code work?
- No!
- Need to instantiate an object before you can use it!

Creating new Objects

- To use an object we need a reference to it
- Use the **new** command to instantiate an object
- What do you think is happening in memory?

```
public static void main(...)  
{  
    Car chevy;  
    Car honda;  
    honda = new Car(10, 20);  
}
```

Creating new Objects

- To use an object we need a reference to it
- Use the **new** command to instantiate an object
 - Reserves memory on the **Heap** for that object class
- Each new object gets its own chunk of memory
 - But they share functions and static class variables

```
public static void main(...)  
{  
    Car chevy;  
    Car honda;  
    honda = new Car(10, 20);  
}
```

Stack

chevy	null
honda	0x1423000

Heap

0x1423000	x	10
0x1423004	y	20



Linked List and Java

- How do we transform C to Java CODE?

```
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;  
}
```

Put code in java/
directory

Class structure and
data members have
been done for you