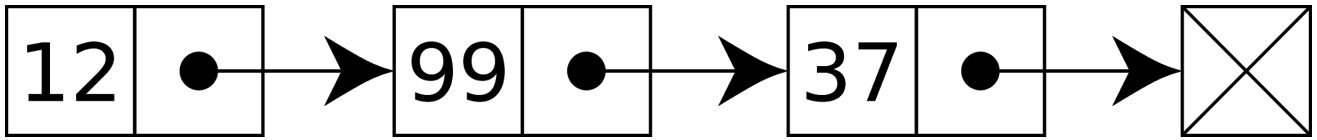
# Creating and Printing Linked Lists

Kevin Browne

# Linked lists

- A linked list is a data structure made up of elements called **nodes** which each **point** to the next node in the list



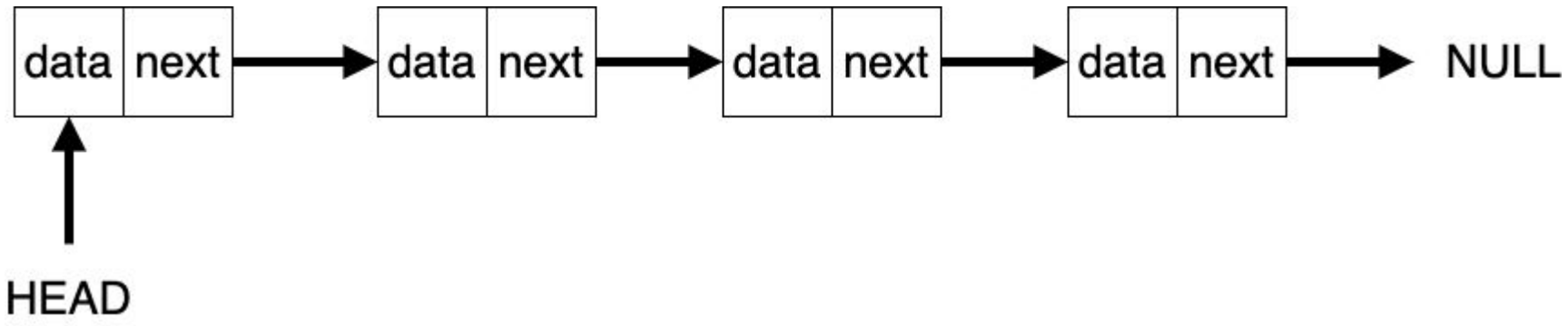

- We call the first node in the list the **head** node, and the last node in the list the **tail** node
  - The last element in the list points to "nothing" (e.g. NULL in C)

Source: https://en.wikipedia.org/wiki/Linked_list#/media/File:Singly-linked-list.svg

# Linked lists in C

- To implement a linked list in C we can use typedef, struct and pointers

- We use typedef and struct to define a **node type** that will store some data (e.g. an int), and that has a member 'next' that is a ***pointer to another node!***



# A linked list node type in C

```
typedef struct node {
    int value;
    struct node *next;
} Node;
```

# A linked list node type in C

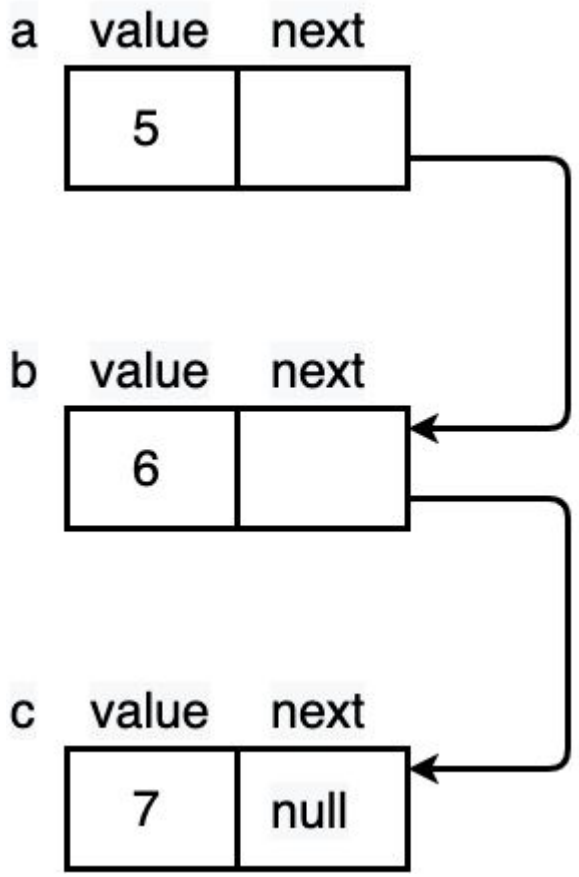
- The "data" associated with each node can be anything we want
- In this case we're keeping it simple as an int value
- But it could be another struct with a set of members, an array, or a set of members within this struct, etc.
- We can self-reference the struct name node to create a pointer to another node

# Linked list on the stack

- Technically there is nothing stopping us from creating and using data structures on the stack
- We can declare and initialize statically allocated nodes with data
- We can have them point to one another in a list by setting the correct next value
- This is not typical for data structures though!

# Creating a linked list on the stack

```
Node a, b, c;
a.value = 5;
b.value = 6;
c.value = 7;
a.next = &b;
b.next = &c;
c.next = NULL;
```



# How can we print a linked list?

- There is no index as with arrays, so what do we do?
  - We ***traverse*** the elements of the data structure by stepping through the elements of the data structure
- Traversing a linked list...
  - We can create a pointer to a node, and set it equal to the head node
  - We can print out the node's value (perhaps with an "index" for ordering), and then set the pointer we created to the next node
  - We can repeat the above step until the next node is null

# Example: printing all list nodes

```
void print_list(Node *head)
{
  Node *current;
  current = head;
  int i = 0;
  while (current != NULL)
  {
    printf("Node %d: %d\n", i, current->value);
    i++;
    current = current->next;
  }
}
```

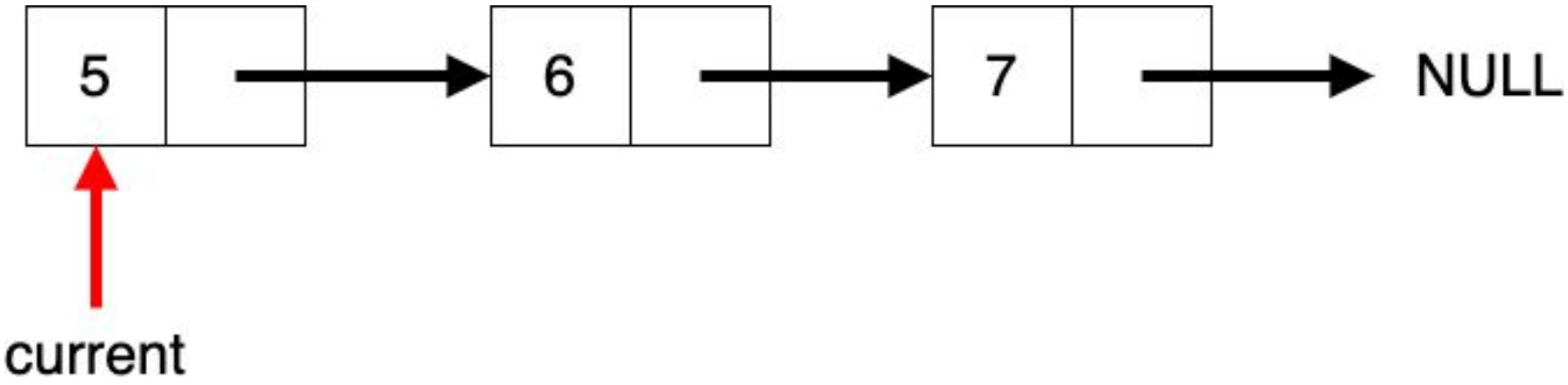
# What is the -> operator?

- The **->** operator functions like the **.** operator for structs, except it works with ***pointers to structs***

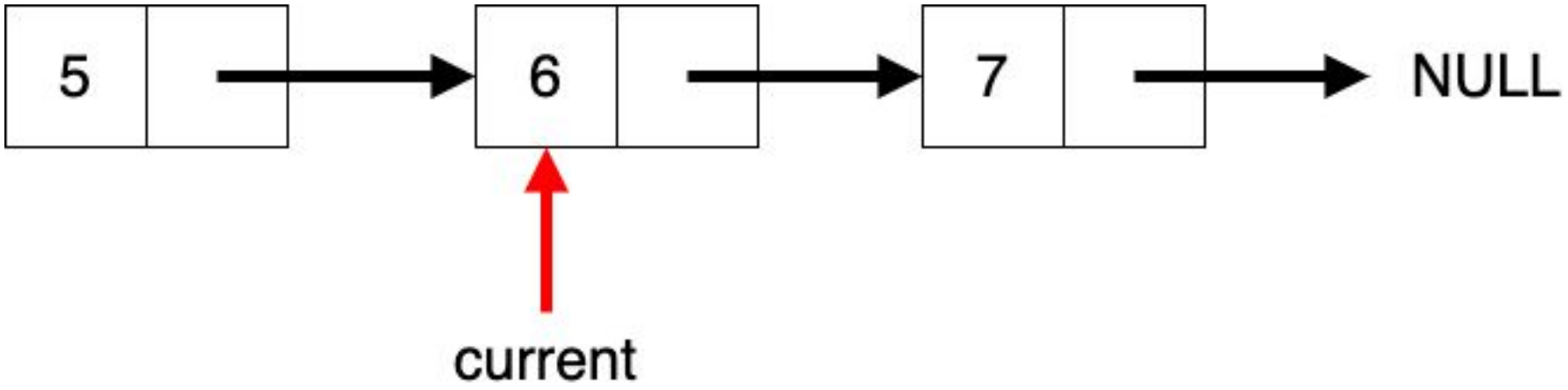
- So **current->value** refers to the member value of the struct that current is pointing to

- So it's the equivalent of **(*current).value**
  - It's a syntactic sugar in that it doesn't do anything we couldn’t do before, but it's more natural to read

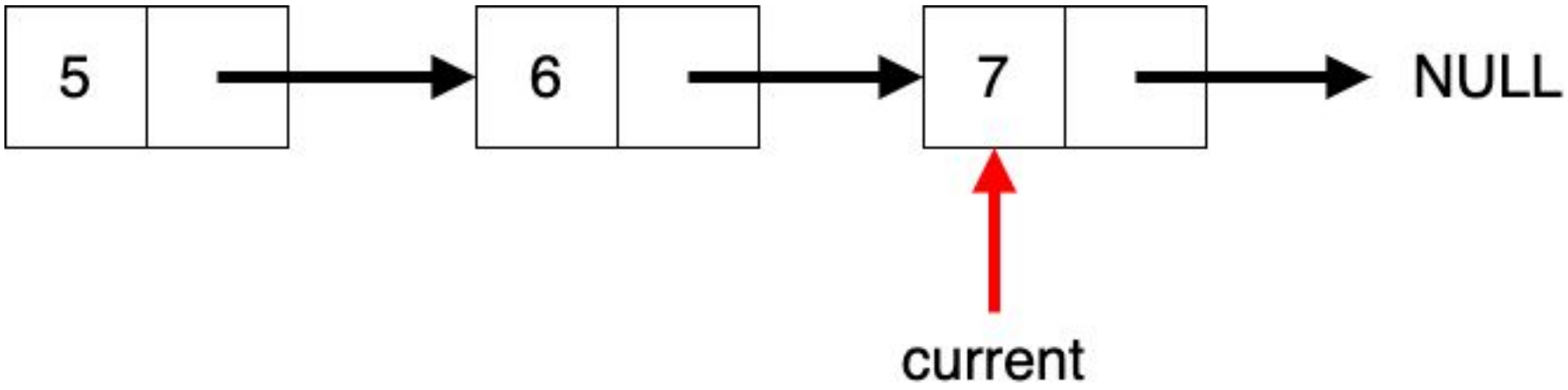
# Traversing the list...



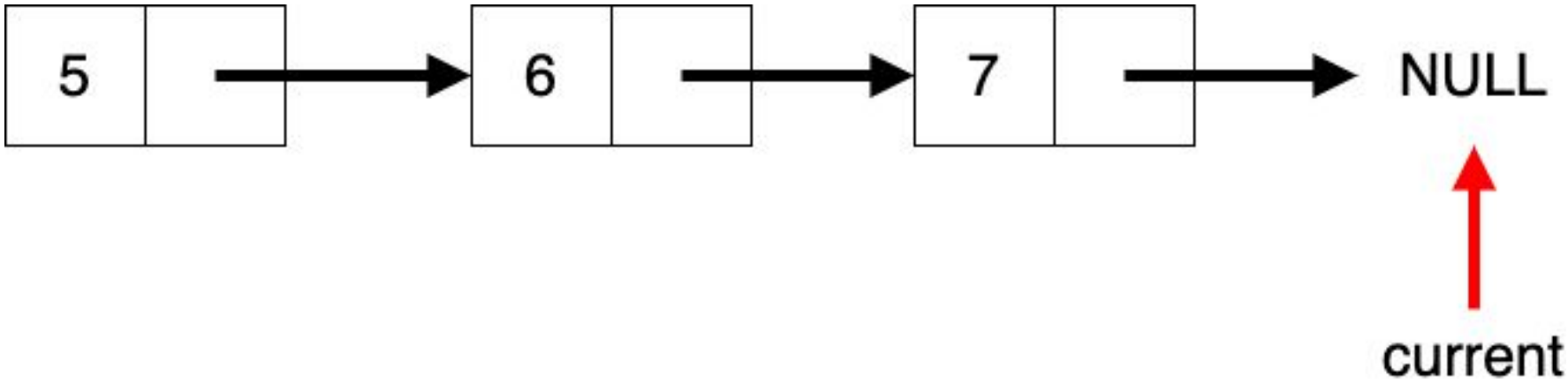
# Traversing the list...



# Traversing the list...



# Traversing the list...



We don't really need to print out the "index" of each node (because there is no index as with arrays), but doing it makes the order more explicit...

```
Print out stack list:
Node 0: 5
Node 1: 6
Node 2: 7
```