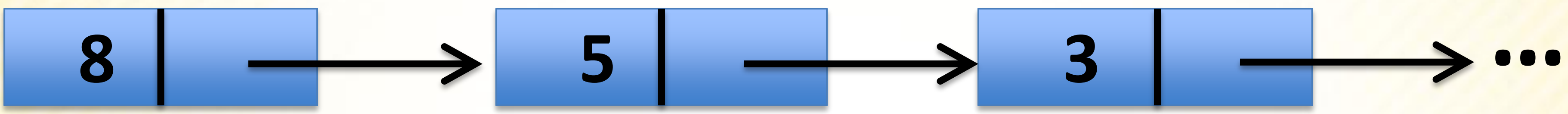
LINKED LIST INTRO

COP 3502

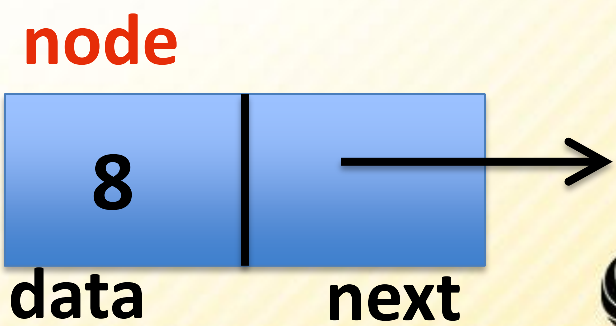
Linked List Introduction

- A **Linked List**
  - Is the simplest form of a linked structure.
  - It consists of a chain of data locations called ***nodes***



- A ***node***
  - Holds a piece of information **AND**
  - a link to the next node

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



Linked List Introduction

- What are Linked Lists?
  - Abstraction of a list
    - that is, a sequence of nodes in which each node is linked to the node following it.
- Why not use an array?
  - Each node in an array is stored in a contiguous space in memory, this means:
    - Arrays are fixed size (not dynamic)
      - We could realloc more space, but this requires work
    - Inserting and deleting elements is difficult
      - For example, in an array of size 100, if we want to insert an element after the 10^{th} element – what do we have to do?
      - We have to shift the remaining 90 elements in some way.

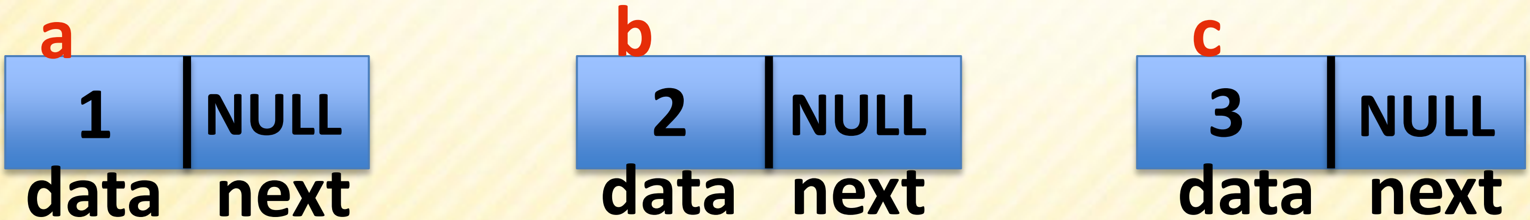
Linked List Introduction

- Pros
  - They are dynamic – so length can increase or decrease as necessary.
  - Each node does not necessarily follow the previous one in memory.
  - Insertion and deletion is cheap
    - Only need to change a few nodes (at most)
- Is there a negative aspect of linked lists?
  - We do not know the address of any individual node
    - So we have to traverse the list to find it, which may take a large # of operations.

Linked List Example

- Let's say we declare 3 Linked List nodes in memory:
  - `struct node a, b, c;`
  - `a.data = 1;`
  - `b.data = 2;`
  - `c.data = 3;`
  - `a.next = b.next = c.next = NULL;`

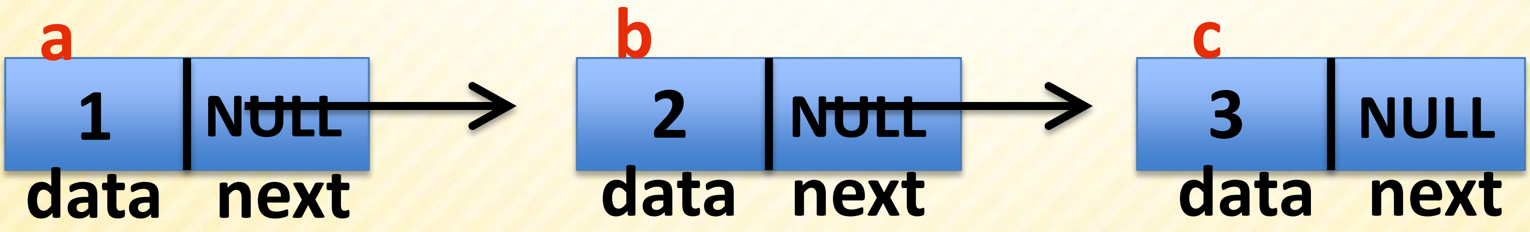
```
struct node {
      int data;
      struct node* next;
};
```



Linked List Example

- Let's say we declare 3 Linked List nodes in memory:
  - `a.next = &b;`
  - `b.next = &c;`
  - `a.next->data` **Has value 2**
  - `a.next->next->data` **Has value 3**
  - `b.next->next->data` **Error!**

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



Linked Lists in Detail

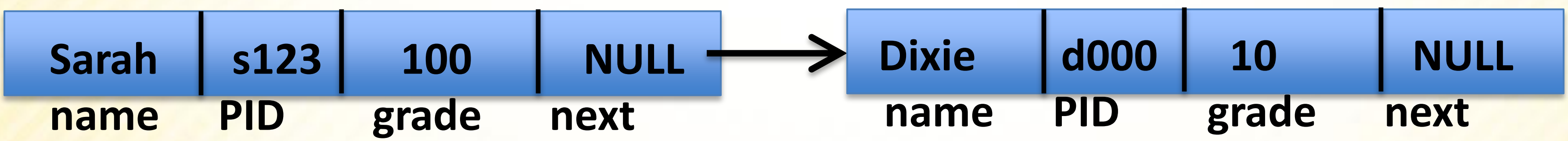
- A linked list is an ordered collection of data
  - Each element (generally called nodes) contains the location of the next element in the list
  - Each node essentially has 2 parts:
  - **The data part**
    - For our examples we're usually just going to use an int, but really we could store anything in each node.
    - If we wanted a linked list of student records we could store PIDs, names, grades, etc.

  - **The link part**
    - This link is used to connect the nodes together.
    - It is just a pointer to the next node in the list.
    - This variable is usually called "next"

Linked Lists

- Node 3 data fields

```
struct node {
      char PID[8];
      char name[80];
      int gradePts;
      struct node* next;
};
```

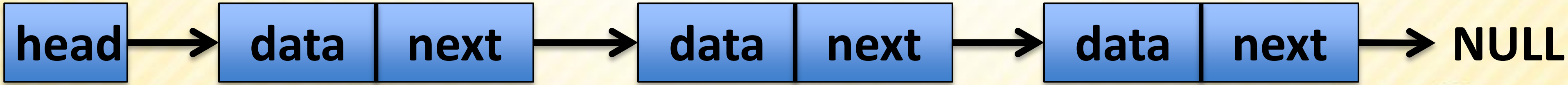


- `struct node s1;`
- `strcpy(s1.name, "Sarah");`
- `strcpy(s1.PID, "s123");`
- `s1.grade = "100";`
- `s1.next = NULL;`

- `struct node s2;`
- `strcpy(s1.name, "Dixie");`
- `strcpy(s1.PID, "d000");`
- `s1.grade = "10";`
- `s1.next = NULL;`
- `s1.next = &s2;`

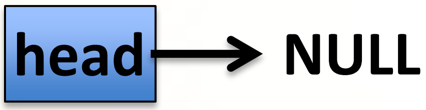
Linked Lists

- How to access nodes of a linked list
  - Each node of the list is created dynamically and points to the next node in the list
    - ➢ So from the first node, we can get to the second, etc.
  - But how do you reach the first node?
    - ➢ You must have a pointer variable that simply points to the front of the list, or the 1st node of the list.
    - ➢ This pointer can be called whatever you want.
      - `head`



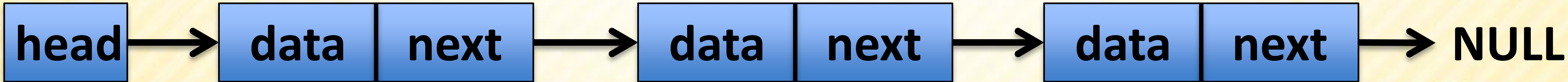
Linked Lists

- Example of an Empty Linked List
  - `struct node* head = NULL;`



Linked Lists

- How to access nodes of a linked list
  - Let's assume we already have a list created with several nodes
    - ➢Don't worry how we made it, we'll cover adding to a list after we cover traversing a list.
  - We access the list via the pointer head
    - ➢How would you move to the 2nd node in the list?



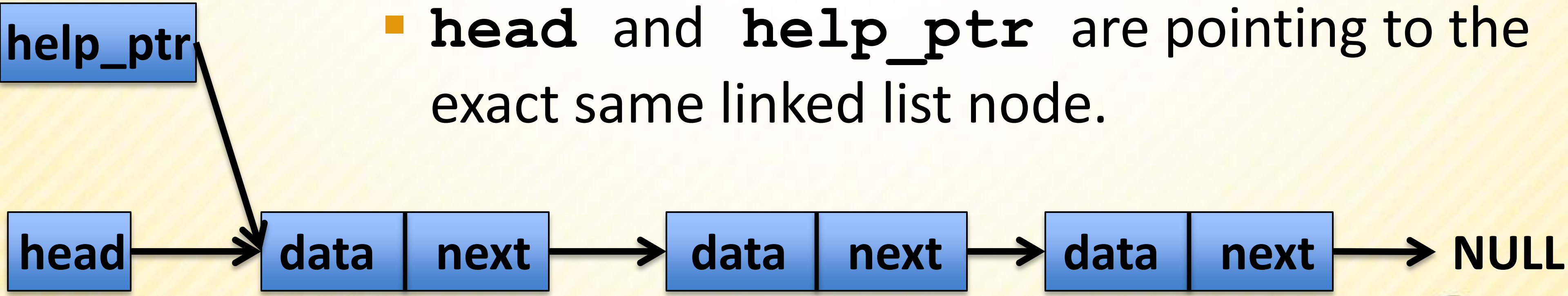
Linked Lists

- How to access nodes of a linked list
  - One of the most common errors is to move the head of the list.
    - if we make the head ptr point to the second node in the list, we would have **NO** way to access the first record.
    - So rather than do that, what we need is a temporary pointer to help us move through the list.

Linked Lists

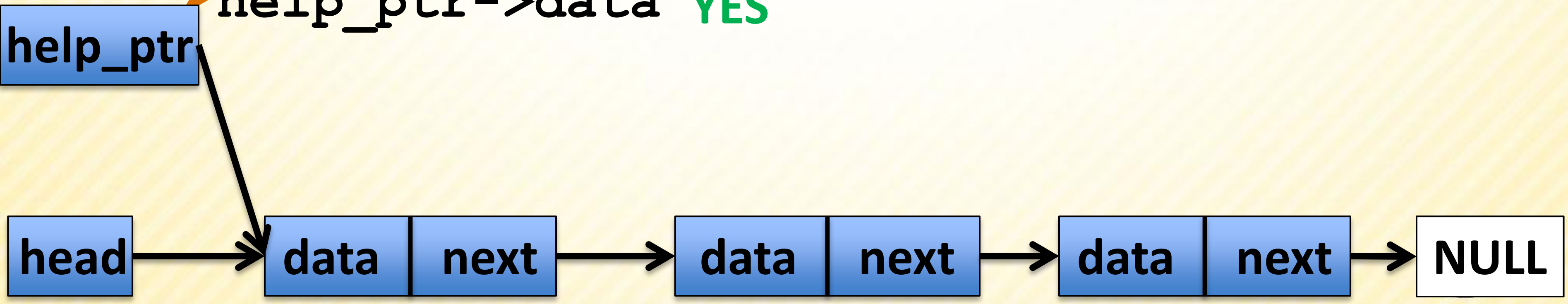
- How to access nodes of a linked list
  - We can define a helper pointer as follows:
  - `struct node *help_ptr;`
  - `help_ptr = head;`

- Something to notice:
  - **`head`** and **`help_ptr`** are pointing to the exact same linked list node.



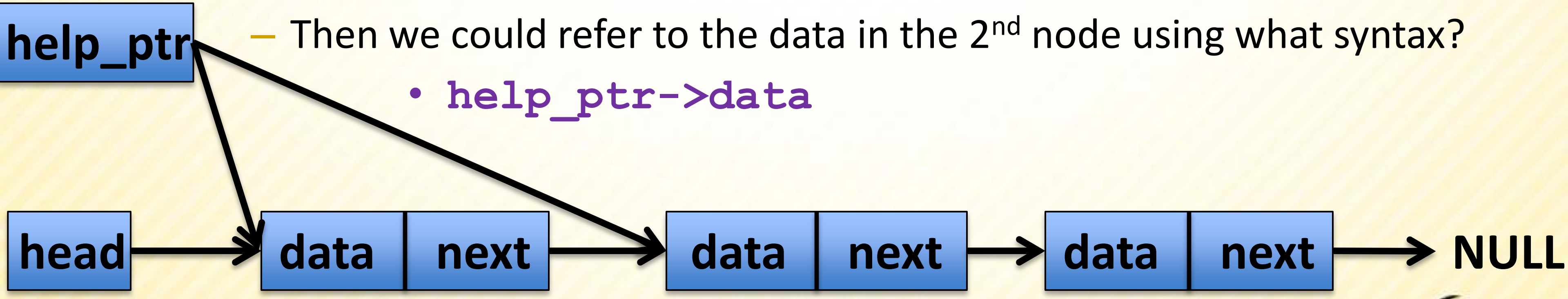
Linked Lists

- How to access nodes of a linked list
  - Another side note, in order to access that first node’s data field, Could we do the following?
    - `head.data` **No, because head is a pointer**
    - `(*head).data` **YES**
    - `(*help_ptr).data` **YES**
    - `head->data` **YES**
    - `help_ptr->data` **YES**



Linked Lists

- How to access nodes of a linked list
  - Now consider using the pointer **`help_ptr`** to traverse the list pointed to by head, we could do something like this:
    - **`help_ptr = help_ptr->next;`**
      - **Note that the syntax is correct because both sides of the statement our pointers to linked lists.**
      - Then we could refer to the data in the 2nd node using what syntax?
        - **`help_ptr->data`**

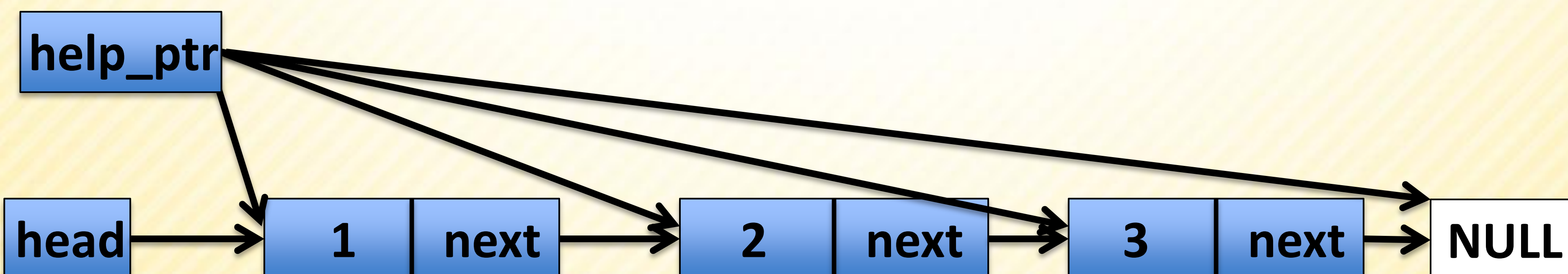


Linked Lists

- Apply this procedure to print a linked list:
  - Assume head is already pointing to a valid list of values

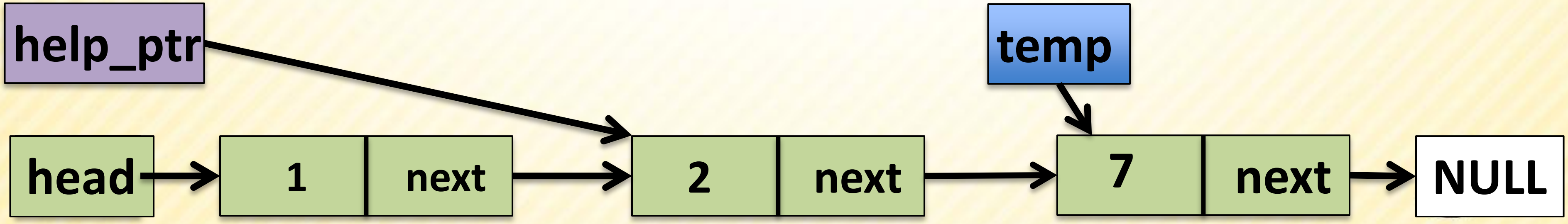
```
struct node *help_ptr;
help_ptr = head;

while (help_ptr != NULL) {
    printf("%d ", help_ptr->data);
    help_ptr = help_ptr->next;
}
```

1 2 3



Linked Lists: How to Add a Node

- This is how to create a node to be added to a list:
  - `struct node *temp;`
  - `temp = (struct node*)malloc(sizeof(struct node));`
  - `temp->data = 7;`
  - `temp->next = NULL;`
- Now to add this node to the end of a list,
  - Assume `help_ptr` is already pointing to the last node in some list.
  - Then all we have to do is connect the node `help_ptr` is pointing to, to `temp`:
    - `help_ptr->next = temp;`



Linked Lists: How to Add a Node

- Now we can create a function that traverses a list and adds a node to the end of the list:

```
struct node* AddEnd(struct node* head, int val) {
    // Create the new node

    // if the list is empty (head == NULL) return
    // the new node

    // Create a helper pointer to traverse the list

    // Traverse the list until the end

    // Add the new node to the end

    // return the front of the list
}
```

```
struct node* AddEnd(struct node* head, int val) {
    // Create the new node




    // if the list is empty (head == NULL) return
    // the new node

    // Create a helper pointer to traverse the list



    // Traverse the list until the end




    // Add the new node to the end

    // return the front of the list
}
```

```
struct node* AddEnd(struct node* head, int val) {
    // Create the new node
    struct node *temp;
    temp = (struct node*)malloc(sizeof(struct node));
    temp->data = val;
    temp->next = NULL;

    // if the list is empty (head == NULL) return
    // the new node

    // Create a helper pointer to traverse the list



    // Traverse the list until the end





    // Add the new node to the end

    // return the front of the list
}
```

```c
struct node* AddEnd(struct node* head, int val) {
    // Create the new node
    struct node *temp;
    temp = (struct node*)malloc(sizeof(struct node));
    temp->data = val;
    temp->next = NULL;

    if (head == NULL) return temp;

    // Create a helper pointer to traverse the list



    // Traverse the list until the end





    // Add the new node to the end

    // return the front of the list
}
```

```c
struct node* AddEnd(struct node* head, int val) {
    // Create the new node
    struct node *temp;
    temp = (struct node*)malloc(sizeof(struct node));
    temp->data = val;
    temp->next = NULL;

    if (head == NULL) return temp;

    // Create a helper pointer to traverse the list
    struct node *curr;
    curr = head;

    // Traverse the list until the end




    // Add the new node to the end

    // return the front of the list
}
```

```c
struct node* AddEnd(struct node* head, int val) {
    // Create the new node
    struct node *temp;
    temp = (struct node*)malloc(sizeof(struct node));
    temp->data = val;
    temp->next = NULL;

    if (head == NULL) return temp;

    // Create a helper pointer to traverse the list
    struct node *curr;
    curr = head;

    // Traverse the list until the end
    while (curr->next != NULL) {
         curr = curr>next;
    }

    // Add the new node to the end

    // return the front of the list
}
```

```c
struct node* AddEnd(struct node* head, int val) {
    // Create the new node
    struct node *temp;
    temp = (struct node*)malloc(sizeof(struct node));
    temp->data = val;
    temp->next = NULL;

    if (head == NULL) return temp;

    // Create a helper pointer to traverse the list
    struct node *curr;
    curr = head;

    // Traverse the list until the end
    while (curr->next != NULL) {
        curr = curr>next;
    }

    curr->next = temp;

    // return the front of the list
}
```

```
struct node* AddEnd(struct node* head, int val) {
    // Create the new node
    struct node *temp;
    temp = (struct node*)malloc(sizeof(struct node));
    temp->data = val;
    temp->next = NULL;

    if (head == NULL) return temp;

    // Create a helper pointer to traverse the list
    struct node *curr;
    curr = head;

    // Traverse the list until the end
    while (curr->next != NULL) {
        curr = curr>next;
    }

    curr->next = temp;

    return head;
}
```

Linked Lists

- Let's show an example of creating a list using the function we just created…
  - shown in class

Linked Lists: How to Add a Node

- Now we can create a function that traverses a list and adds a node to the end of the list:

```
struct node* AddEnd(struct node* head, int val) {
    struct node *temp;
    temp = (struct node*)malloc(sizeof(struct node));
    temp->data = val;
    temp->next = NULL;

    if (head == NULL) return temp;

    struct node *curr;
    curr = head;

    while (curr->next!= NULL) {
        curr = curr>next;
    }
```