COMP1511 - Programming Fundamentals

Week 8 - Lecture 13

What did we learn last week?

Structs and Memory

- Our own custom variable types made up of other variables
- Allocating memory for use beyond the scope of functions

Multiple File Projects

- Separating code into different files

Linked Lists

- structs, pointers and memory allocation all together!

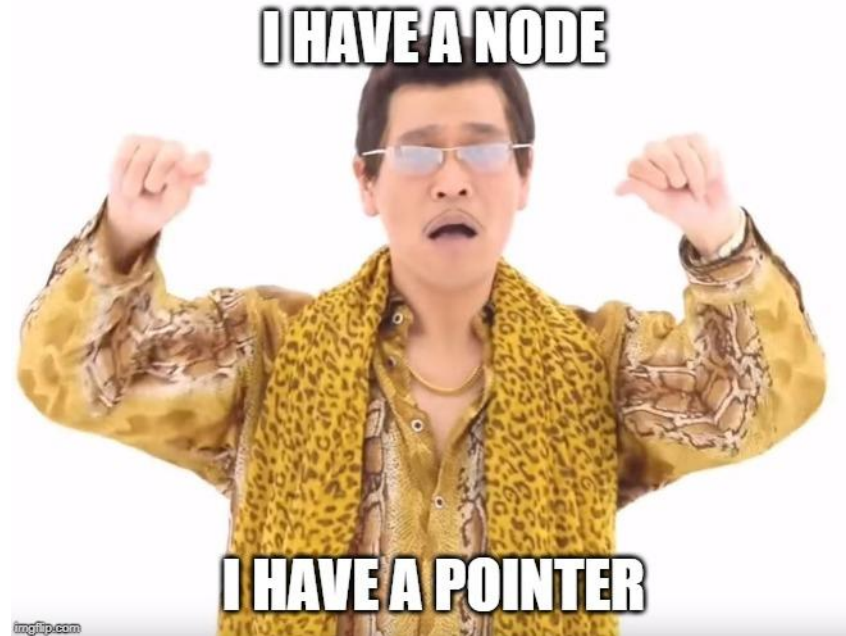# What are we learning today?

Linked Lists

- Continuing our work from last week
- Continuing our example of a Linked List project
- Adding to Linked Lists
- Searching through a list for specific conditions

Recap - Linked Lists

A chain of identical structs to hold information
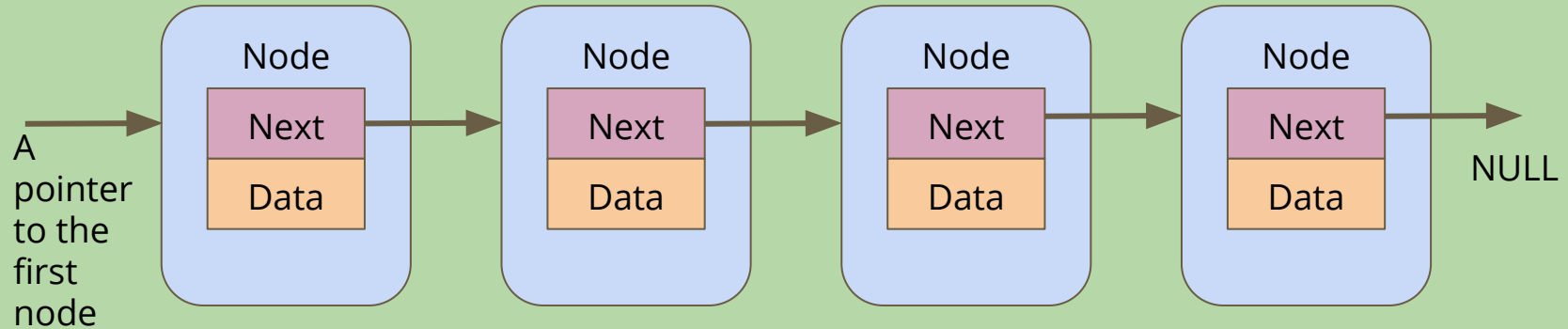
- Pointers to the same type of struct so they can be chained together
- Some kind of information stored in the struct

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

A Linked List

A program's memory (not to scale)

A pointer to the first node

Node — Next — Data

Node — Next — Data

Node — Next — Data

Node — Next — Data

NULL

Looping through a Linked List

Loop by using the next pointer

- We can jump to the next node by following the current node's next pointer
- We know we're at the end if the next pointer is NULL

```c
// Loop through a list of nodes, printing out their data
void printData(struct node *n) {
    while (n != NULL) {
        printf("%d\n", n->data);
        n = n->next;
    }
}
```

Battle Royale - Continued

What do we have so far?

- We've defined basic player structs (linked list nodes)
- We have a function to allocate and create them
- A way of building a list that just uses the create function
- A function to loop through a list and print out the names

Player nodes

A basic linked list node struct

```c
struct player {
    char name[MAX_NAME_LENGTH];
    struct player *next;
};
```

Creating players

A function that creates a node

```c
// Create a player node using the name and next pointer provided
// Return a pointer to this node
struct player *createPlayer(char *newName, struct player *newNext) {
    struct player *newPlayer = malloc(sizeof (struct player));
    strcpy(newPlayer->name, newName);
    newPlayer->next = newNext;
    return newPlayer;
}
```

Creating the list

This is a simple way of doing this

```c
int main(void) {
    // create the list of players
    struct node *head = NULL;
    head = createPlayer("Marc", head);
    head = createPlayer("Chicken", head);
    head = createPlayer("Aang", head);
    head = createPlayer("Katara", head);

    return 0;
}
```

This method basically adds a new element to the start of the list each time

Printing out the list of players

Looping through and printing out the name of each player

- Starting with the pointer to the head of the list
- Use whatever data is inside the player node
- Then move the curr pointer to the next node
- If the curr pointer is NULL, then we've reached the end of the list

```c
// Loop through the list and print out the player names
void printPlayers(struct player *playerList) {
    struct player *curr = playerList;
    while (curr != NULL) {
        printf("%s\n", curr->name);
        curr = curr->next;
    }
}
```

Battle Royale - What's next?

What else does the program need?

- Add players to the game
  - Inserting into a list
- Maintain a list of players that's in order
  - Inserting into a specific position in a list

Inserting Nodes into a Linked List

Linked Lists allow you to insert nodes in between other nodes

- We can do this by simply aiming next pointers to the right places
- We find two linked nodes that we want to put a node between
- We take the **next** of the first node and point it at our new node
- We take the **next** of the new node and point it at the second node

This is much less complicated with diagrams . . .
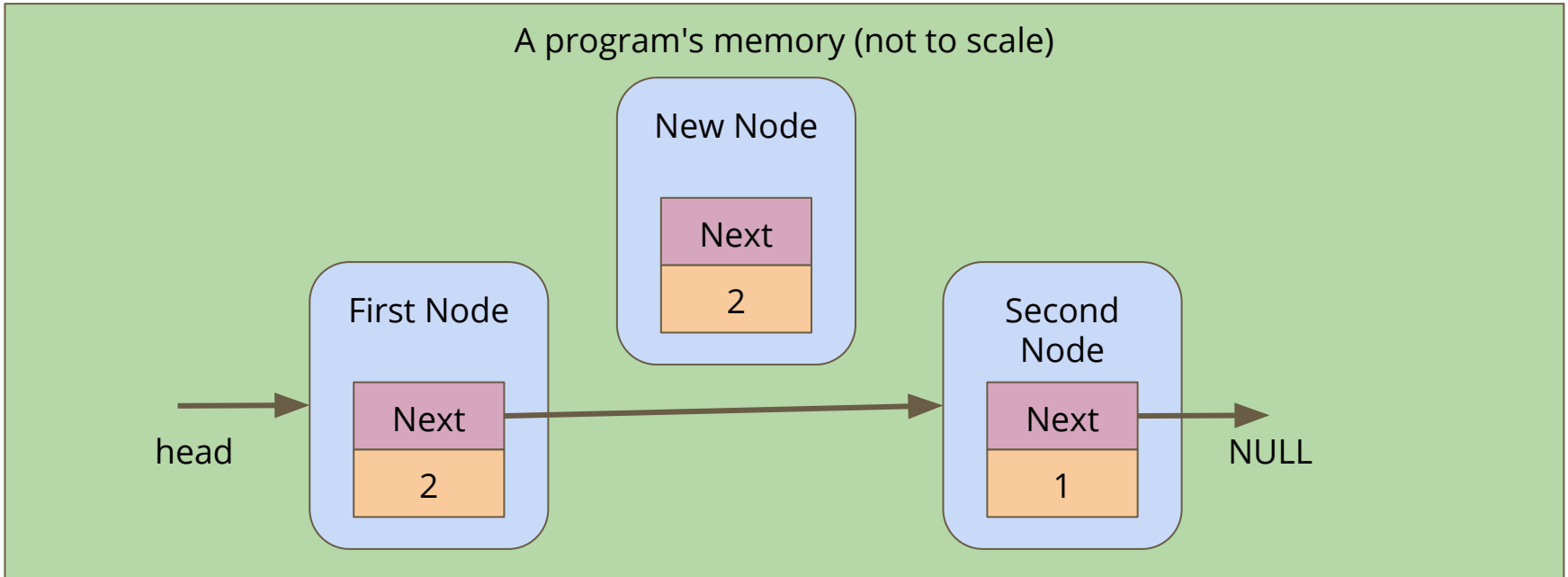
Our Linked List

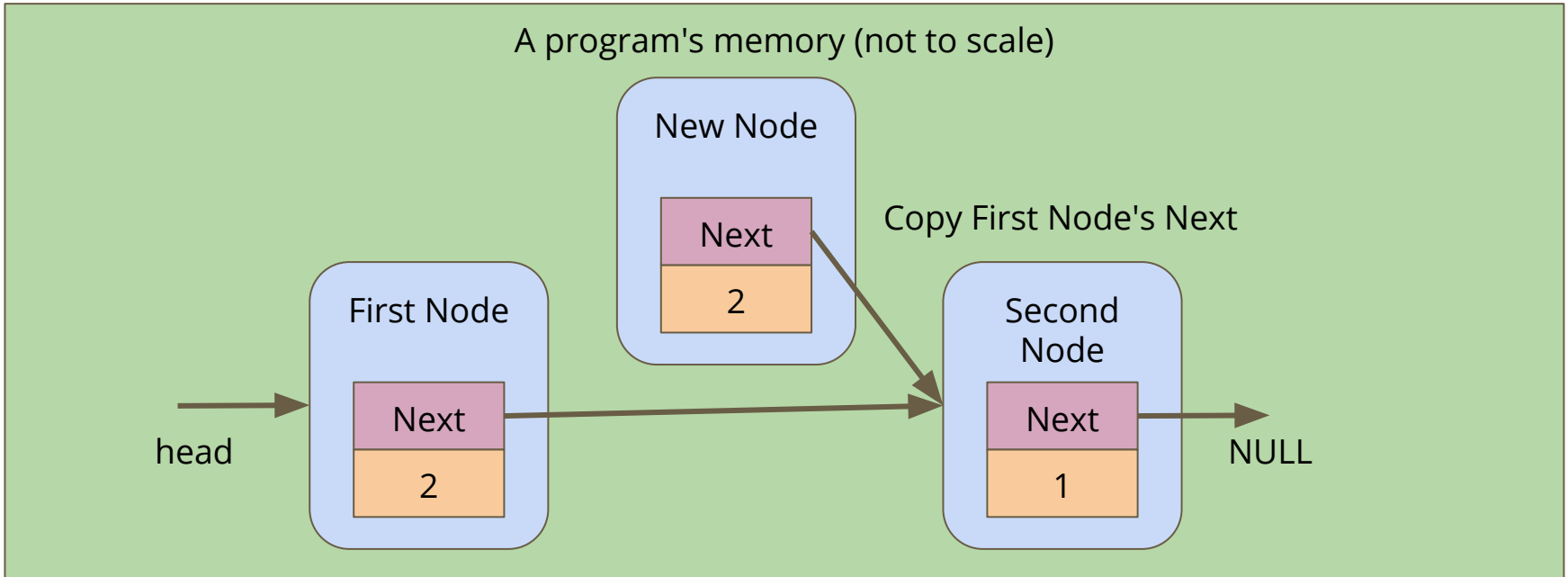Before we've tried to insert anything

Create a node

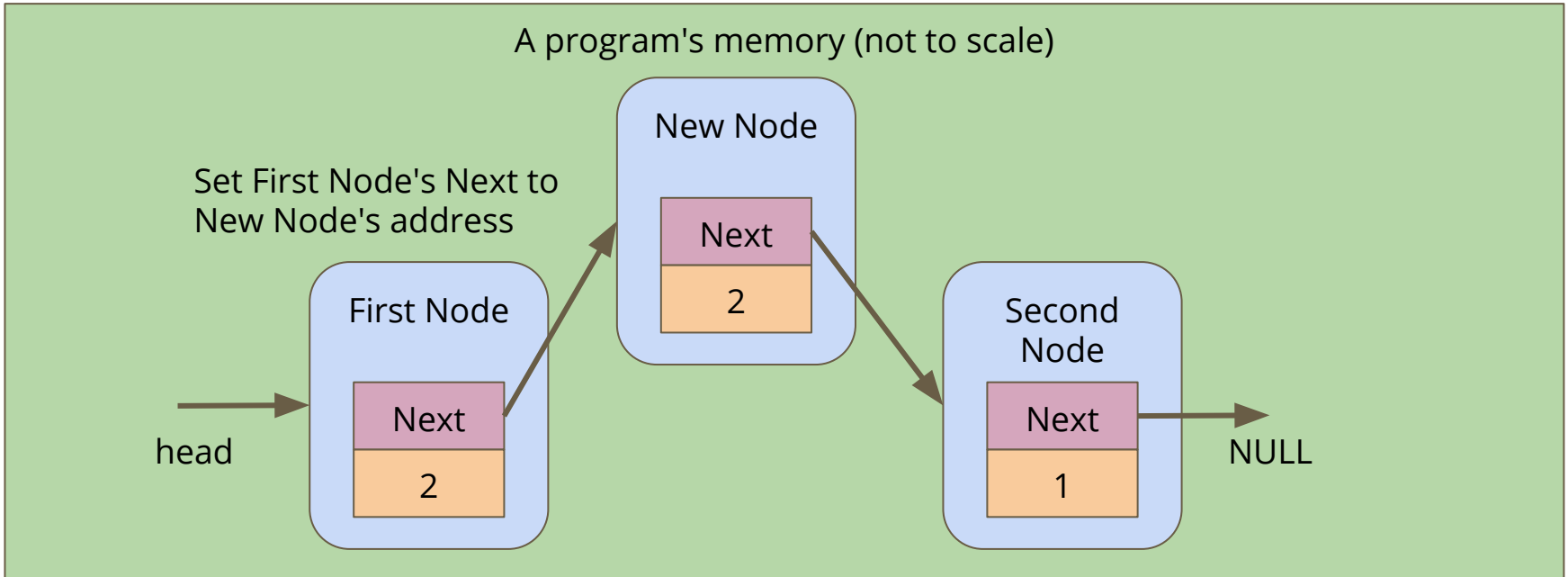A new node is made, it's not connected to anything yet

Connect the new node to the second node

Alter the **next** pointer on the New Node

Connect the first node to the new node

Alter the **next** pointer on the First Node

Break Time

Homework - it's not real homework, just things that can inspire you

- *AlphaGo* Documentary (on Netflix)
- *I, Robot* Short Stories (Isaac Asimov)
- *Snow Crash* and *The Cryptonomicon* Novels (Neal Stephenson)
- *Human Resource Machine* Game (on Steam, iOS and Android)
- *Space Alert* Board Game (Vlaada Chvatil)

Code for insertion of players

```c
// Create and insert a new node into a list after a given insert position
struct player *insertAfter(struct player* insertPos, char newName[]) {
    struct player *p = createPlayer(newName, NULL);
    if (insertPos == NULL) {
        // List is empty, p becomes the only element in the list
        insertPos = p;
        p->next = NULL;
    } else {
        // Set the new player (p)'s next to after the insertion position
        p->next = insertPos->next;
        // Set the insert position node's next to now aim at p
        insertPos->next = p;
    }
    return insertPos;
}
```

Inserting Players to create a list

We can use insertion to have greater control of where players end up

In this example, Chicken is inserted after the head (Marc), then Aang is also inserted after Marc (and before Chicken)

```c
int main(void) {
    // create the list of players
    struct node *head = createPlayer("Marc", NULL);
    insertAfter("Chicken", head);
    insertAfter("Aang", head);

    printPlayers(head);

    return 0;
}
```

Insertion with some conditions

We can now insert into any position in a Linked List

- We can read the data in a node and decide whether we want to insert before or after it
- Let's insert our elements into our list based on alphabetical order
- We're going to use a `string.h` function, `strcmp()` for this
- `strcmp()` compares two strings, and returns
  - 0 if they're equal
  - negative if the first has a lower ascii value than the second
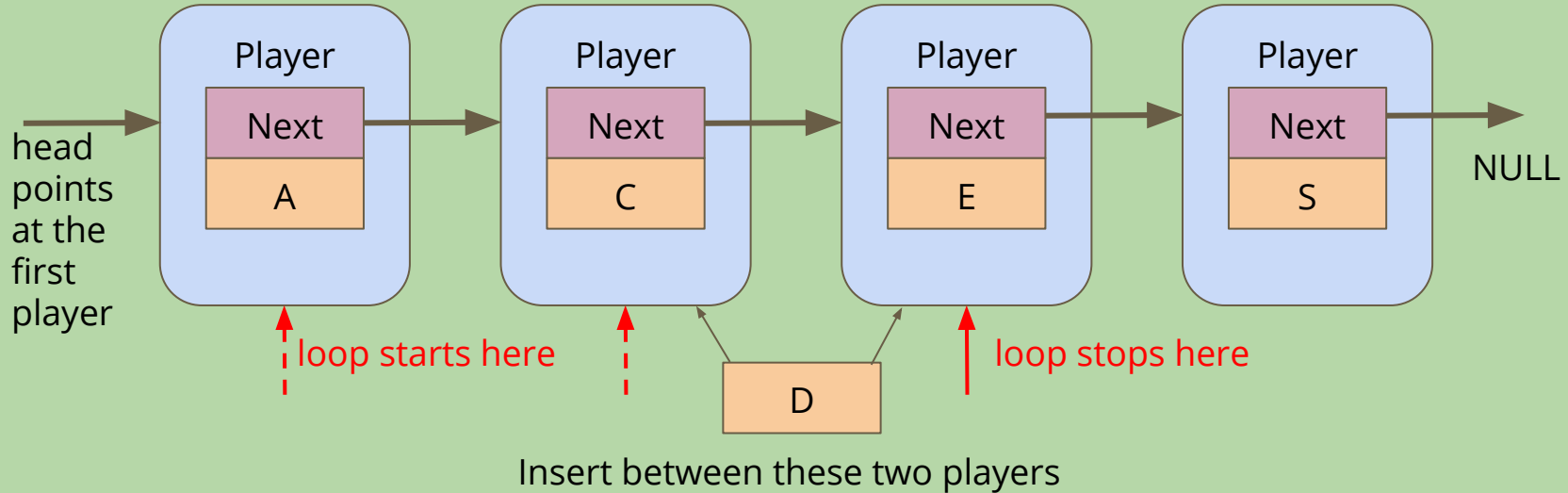 - positive if the first has a higher ascii value than the second

Finding where to insert

We're going to loop through the list

- This loop assumes the list is already in alphabetical order
- Each time we loop, we're going to keep track of the previous player
- We'll test the name of each player using `strcmp()`
- We stop looping once we find the first name that's "higher" than ours
- Then we insert before that player

Finding the insertion point

Attempting to insert a player with name: "D" into a sorted list while maintaining the alphabetical order

head points at the first player

Player	Player	Player	Player
Next	Next	Next	Next
A	C	E	S

NULL

loop starts here

loop stops here

D

Insert between these two players

Inserting into a list Alphabetically

```c
struct player *insertAlphabetical(char newName[], struct player* head) {
    struct player *previous = NULL;
    struct player *p = head;
    // Loop through the list and find the right place for the new name
    while (p != NULL && strcmp(newName, p->name) > 0) {
        previous = p;
        p = p->next;
    }
    struct player *insertionPoint = insertAfter(newName, previous);
    // Return the head of the list (even if it has changed)
    if (previous == NULL) { // we inserted at the start of the list
        insertionPoint->next = p;
        return insertionPoint;
    } else {
        return head;
    }
}
```

What did we learn today?

Linked Lists

- Recap of Linked Lists
- Building the list
- Looping through the list
- Inserting nodes at a specific location
- Inserting nodes into an ordered list