Recursion -- Solving Problems with your Mind...

And some C code...
“Isn’t everything solving problems with your mind?”

~Marc Chee, 2021
What are we learning about today

1. A new way of thinking about solving problems.
2. A way to write shorter but elegant programs.
How have we learned to solve a problem so far?

**Problem:** Take a linked list of positive integers, and find the largest number.
How have we learned to solve a problem so far?

Problem: Take a linked list of positive integers, and find the largest number.

Solution:
- Initialize a variable called \texttt{max} to something really small (-1)
- Loop through each node of the linked list.
  - If \texttt{curr->value} is bigger than \texttt{max}, update \texttt{max}

This is a perfectly fine way to solve this problem!
max = -1

```c
int find_max(struct node *head) {
    int max = -1;
    struct node *curr = head;

    while (curr != NULL) {
        if (max < curr->value) {
            max = curr->value;
        }
        curr = curr->next;
    }
    return max;
}
```
max = 42 (was -1)

```c
int find_max(struct node *head) {
    int max = -1;
    struct node *curr = head;

    while (curr != NULL) {
        if (max < curr->value) {
            max = curr->value;
        }
        curr = curr->next;
    }
    return max;
}
```
max = 42

```c
int find_max(struct node *head) {
    int max = -1;
    struct node *curr = head;

    while (curr != NULL) {
        if (max < curr->value) {
            max = curr->value;
        }
        curr = curr->next;
    }
    return max;
}
```
max = 100 (was 42)

```c
// Find the maximum value in the linked list
int find_max(struct node *head) {
    int max = -1;
    struct node *curr = head;

    while (curr != NULL) {
        if (max < curr->value) {
            max = curr->value;
        }
        curr = curr->next;
    }
    return max;
}
```
```c
int find_max(struct node *head) {
    int max = -1;
    struct node *curr = head;

    while (curr != NULL) {
        if (max < curr->value) {
            max = curr->value;
        }
        curr = curr->next;
    }

    return max;
}
```
```c
// Find the maximum value in the linked list
int find_max(struct node *head) {
    int max = -1;
    struct node *curr = head;

    while (curr != NULL) {
        if (max < curr->value) {
            max = curr->value;
        }
        curr = curr->next;
    }
    return max;
}
```
But now... a new way!

**Problem:** Take a linked list of positive integers, and find the largest number.

**Observation:** If we have a linked list with at least one node in it, there are two possibilities:

1) The largest number is the first number.
2) The largest number is something **other** than the first number.
Possibility #1: Largest number is the current one

42 → 17 → 12 → 1 → NULL
Possibility #2: Largest number is in the rest of the list
So what?

Problem: Take a linked list of positive integers, and find the largest number.

Solution:
- Find the maximum number of everything after the current node
- Either:
  - Everything in the rest of the list is smaller than the current element.
  - Or, something in the rest of the list is bigger than the current element.
Functions call themselves?!?!?!

```c
// Find the maximum value in the linked list
int find_max(struct node *head) {
    int maximum_so_far = find_max(head->next);
    if (maximum_so_far < head->value) {
        // we have found a new node that's bigger
        maximum_so_far = head->value;
    }

    return maximum_so_far;
}
```
maximum_so_far = the max of the rest of the list

// Find the maximum value in the linked list
int find_max(struct node *head) {
    int maximum_so_far = find_max(head->next);
    if (maximum_so_far < head->value) {
        // we have found a new node that's bigger
        maximum_so_far = head->value;
    }

    return maximum_so_far;
}
maximum_so_far = the max of the rest of the list

// Find the maximum value in the linked list
int find_max(struct node *head) {
    int maximum_so_far = find_max(head->next);
    if (maximum_so_far < head->value) {
        // we have found a new node that's bigger
        maximum_so_far = head->value;
    }
    return maximum_so_far;
}
maximum_so_far = the max of the rest of the list

```c
// Find the maximum value in the linked list
int find_max(struct node *head) {
    int maximum_so_far = find_max(head->next);
    if (maximum_so_far < head->value) {
        // we have found a new node that's bigger
        maximum_so_far = head->value;
    }
    return maximum_so_far;
}
```
maximum_so_far = the max of the rest of the list

```c
void find_max(struct node *head) {
    int maximum_so_far = find_max(head->next);
    if (maximum_so_far < head->value) {
        // we have found a new node that's bigger
        maximum_so_far = head->value;
    }
    return maximum_so_far;
}
```
// Find the maximum value in the linked list
int find_max(struct node *head) {
    int maximum_so_far = find_max(head->next);
    if (maximum_so_far < head->value) {
        // we have found a new node that's bigger
        maximum_so_far = head->value;
    }
    return maximum_so_far;
}
Uh Oh...

/tmp/starter_code.c:18:41: runtime error - accessing a field via a NULL pointer

dcc explanation: You are using a pointer which is NULL
   A common error is using p->field when p == NULL.

Execution stopped in find_max(head=NULL) in /tmp/starter_code.c at line 18:

int find_max(struct node *head) {
   -- int maximum_so_far = find_max(head->next);
   if (maximum_so_far < head->value) {
      // we have found a new node that's bigger
      maximum_so_far = head->value;

Values when execution stopped:

head = NULL
maximum_so_far = <uninitialized value>

Function Call Traceback
find_max(head=NULL) called at line 37 of /tmp/starter_code.c
main()
We should have known...

```c
starter_code.c:17:33: warning: all paths through this function will call itself [-Wfinite-recursion]
int find_max(struct node *head) {
  
```
One problem...

What do we do when we get to the end of the list? Currently we have “infinite recursion”.

So we need to include a special case: “what to do when we have no elements left”. We call this the base case.

**Solution:** Return a negative number (-1) if given an empty list. Since our list always contains positive integers, it’s smaller than anything else.
Solution, Attempt 2 [this time it’ll work [hopefully]]

Problem: Take a linked list, and find the largest number.

Solution:

- Find the maximum value in nodes after the current node.
- Either:
  - We have an empty list, in which case return -1.
  - Everything in the rest of the list is smaller than the current element.
  - Or, something in the rest of the list is bigger than the current element.
maximum_so_far = the max of the rest of the list

```c
#define NO_MAXIMUM -1

// Find the maximum value in the linked list
int find_max(struct node *head) {
    if (head == NULL) {
        return NO_MAXIMUM;
    }

    int maximum_so_far = find_max(head->next);
    if (maximum_so_far < head->value) {
        // we have found a new node that's bigger
        maximum_so_far = head->value;
    }

    return maximum_so_far;
}
```
maximum_so_far = the max of the rest of the list

```c
#define NO_MAXIMUM -1

// Find the maximum value in the linked list
int find_max(struct node *head) {
    if (head == NULL) {
        return NO_MAXIMUM;
    }
    int maximum_so_far = find_max(head->next);
    if (maximum_so_far < head->value) {  // we have found a new node that's bigger
        maximum_so_far = head->value;
    }
    return maximum_so_far;
}
```
# define NO_MAXIMUM -1

// Find the maximum value in the linked list
int find_max(struct node *head) {
    if (head == NULL) {
        return NO_MAXIMUM;
    }
    int maximum_so_far = find_max(head->next);
    if (maximum_so_far < head->value) {
        // we have found a new node that's bigger
        maximum_so_far = head->value;
    }
    return maximum_so_far;
}
#define NO_MAXIMUM -1

// Find the maximum value in the linked list
int find_max(struct node *head) {
  if (head == NULL) {
    return NO_MAXIMUM;
  }
  int maximum_so_far = find_max(head->next);
  if (maximum_so_far < head->value) {
    // we have found a new node that's bigger
    maximum_so_far = head->value;
  }
  return maximum_so_far;
}
#define NO_MAXIMUM -1

// Find the maximum value in the linked list
int find_max(struct node *head) {
    if (head == NULL) {
        return NO_MAXIMUM;
    }
    int maximum_so_far = find_max(head->next);
    if (maximum_so_far < head->value) {
        // we have found a new node that's bigger
        maximum_so_far = head->value;
    }
    return maximum_so_far;
}
#define NO_MAXIMUM -1

// Find the maximum value in the linked list
int find_max(struct node *head) {
    if (head == NULL) {
        return NO_MAXIMUM;
    }
    int maximum_so_far = find_max(head->next);
    if (maximum_so_far < head->value) {
        // we have found a new node that's bigger
        maximum_so_far = head->value;
    }
    return maximum_so_far;
}
maximum_so_far = 100 or 1

```c
#define NO_MAXIMUM -1

// Find the maximum value in the linked list
int find_max(struct node *head) {
    if (head == NULL) {
        return NO_MAXIMUM;
    }
    int maximum_so_far = find_max(head->next);
    if (maximum_so_far < head->value) {
        // we have found a new node that's bigger
        maximum_so_far = head->value;
    }
    return maximum_so_far;
}
```
# Navigate through the list to find the maximum value

```c
#define NO_MAXIMUM -1

int find_max(struct node *head) {
    if (head == NULL) {
        return NO_MAXIMUM;
    }
    int maximum_so_far = find_max(head->next);
    if (maximum_so_far < head->value) {
        // we have found a new node that's bigger
        maximum_so_far = head->value;
    }
    return maximum_so_far;
}
```

maximum_so_far = 17 or 100
# Find the maximum value in the linked list

```c
int find_max(struct node *head) {
    if (head == NULL) {
        return NO_MAXIMUM;
    }
    int maximum_so_far = find_max(head->next);
    if (maximum_so_far < head->value) {
        // we have found a new node that's bigger
        maximum_so_far = head->value;
    }
    return maximum_so_far;
}
```

```
define NO_MAXIMUM -1
```

maximum_so_far = 42 or 100
#define NO_MAXIMUM -1

// Find the maximum value in the linked list
int find_max(struct node *head) {
    if (head == NULL) {
        return NO_MAXIMUM;
    }
    int maximum_so_far = find_max(head->next);
    if (maximum_so_far < head->value) {
        // we have found a new node that's bigger
        maximum_so_far = head->value;
    }
    return maximum_so_far;
}
Let's take a look at some real code...
Why is Recursion different from a While Loop

A normal while loop solves your problem by a series of steps that ends up with the right answer.

Recursion solves your problem by solving a stack of smaller problems. Then, it deals with the trivially easy case.
main()
maximum_so_far = the max of the rest of the list

find_max() [42 is at head of list]

main()
**maximum_so_far** = **the max of the rest of the list**

```
find_max() [17 is at head of list]
find_max() [42 is at head of list]
main()
```
maximum_so_far = the max of the rest of the list

find_max() [100 is at head of list]
find_max() [17 is at head of list]
find_max() [42 is at head of list]
main()
maximum_so_far = the max of the rest of the list

find_max() [1 is at head of list]
find_max() [100 is at head of list]
find_max() [17 is at head of list]
find_max() [42 is at head of list]
main()
maximum_so_far = -1

find_max() [NULL is at head of list]
find_max() [1 is at head of list]
find_max() [100 is at head of list]
find_max() [17 is at head of list]
find_max() [42 is at head of list]
main()
maximum_so_far = 1 or -1

find_max() [1 is at head of list]
find_max() [100 is at head of list]
find_max() [17 is at head of list]
find_max() [42 is at head of list]
main()
maximum_so_far = 100 or 1

```
find_max() [100 is at head of list]
find_max() [17 is at head of list]
find_max() [42 is at head of list]
main()
```
maximum_so_far = 17 or 100

find_max() [17 is at head of list]
find_max() [42 is at head of list]
main()
maximum_so_far = 42 or 100

find_max() [42 is at head of list]

main()
main()

maximum_so_far = 100
Break Time...

Why is recursion so cool?

- Functions are incredibly powerful - in fact, they can do anything a w
- Turns out, we don’t even need variables at all -- we can just use functions
  and never modify our variables
- “Functional Programming” is a different style of programming where you
  never assign to variables; and everything is a function.
- The most common example of functional programming is...
Benefits of Recursion -- Printing Forwards and Backwards
Benefits of Recursion -- No Need for a Prev Pointer
Let’s look at a more interesting example...

Let’s say we have a list; can we figure out every way to add it’s nodes together?

![Diagram of a list with nodes 1, 4, 7, and NULL.]

- $0 + 0 + 0 = 0$
- $1 + 0 + 0 = 1$
- $0 + 4 + 0 = 4$
- $1 + 4 + 0 = 5$
- $0 + 0 + 7 = 7$
- $1 + 4 + 7 = 11$
What did we learn today?

1. A new way of thinking about solving problems.
   a. Instead of solving big problems, solve smaller, easier problems and build them up!

2. A way to write shorter but elegant programs.
   a. Recursion can often be neater than using while loops, at the cost of requiring more time to think through.