Remind us where we left off last time?

- Who can remind us what we talked about?

Today Recursion

- Today’s focus: recursion
  - Some people think it’s scary—but it’s not
  - Wonderful tool
    - Makes many tasks easier
    - Some languages only have recursion (no loops)

- Basic idea:
  - Use function you are writing to solve smaller instance of same problem
  - After all, the function you are writing solves that problem!

Structural Recursion

- We have seen two recursively defined data types:
  - Linked Lists
  - Binary Search Trees

- Can think of recursion here in terms of structural recursion
  - Recursively defined data structure
  - Leads to recursive algorithm
  - Examine structural cases
  - Use recursion to attack smaller instances of the same problem
**Structural Recursion**

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

Structurally, an `ll_node *` is either
- NULL
- or
  - pointing at (data + next)
  - next (being an `ll_node *`) is a (smaller) list
  - -> thus suitable for structural recursion

**Consider reversing a list...**

---

**Want to reverse this list...**

What if we already had the rest of the list reversed?

---

Could just add 3 to the back

---

```
ll_node * reverse(ll_node * lst) {
  if (lst == NULL) {
    return NULL;
  } else {
    reverse(lst->next);
    lst->next = NULL;
    return lst;
  }
}
```
Reverse

ll_node * reverse(ll_node * lst) {
    if (lst == NULL) {
        return NULL;
    } else {
        ll_node * revList = reverse(lst->next);
        return add_to_back(lst->data, revList);
    }
}

ECE 590.01 (Hilton): Recursion 13

How to reverse empty list?
Note: cannot recurse here (don't have a recursive structure)

ECE 590.01 (Hilton): Recursion 14

How to reverse empty list?
Just the empty list

ECE 590.01 (Hilton): Recursion 15

In the other case,
Said it would be nice to have Reversal of rest of list

ECE 590.01 (Hilton): Recursion 16

How efficient is this? (Big-O?)

ECE 590.01 (Hilton): Recursion 17

Note: this does a functional update (builds a new list that is the reverse of the original, leaving the original unchanged).
We can do better

- That was O(N²)
- We can reverse a list in O(N) time.
  - We just have to build up our answer on the way down...
  - What we just saw is called "head recursion"
  - Another approach "tail recursion"
    - Return recursive result w/o doing any other work

Reverse: tail recursion

```c
ll_node * reverse(ll_node * lst, ll_node * ans) {
    if (lst == NULL) {
        return ans;
    }
    else {
        return reverse(lst->next, add_to_front(lst->data, ans));
    }
}
```

Now, build up answer on the way down

Still two cases

So now there is nothing else to reverse,
And we just return what we built up so far.
Reverse: tail recursion

```c
ll_node * reverse(ll_node * lst, ll_node * ans) {
    if (lst == NULL) {
        return ans; // This is now O(N)
    } else {
        return reverse(lst->next, add_to_front(list->data, ans));
    }
}
```

ECE 590.01 (Hilton): Recursion

25

Tail recursion = while loops

- The above is not a coincidence:
  + Tail recursion is identical to a while loop (unless your compiler isn't smart enough to do the right transform).

```c
foo(a, b, c) {
    while(cond) {
        a = f(a, b, c);
        b = g(a, b, c);
        c = h(a, b, c);
    }
    return k(a, b, c);
}
```

ECE 590.01 (Hilton): Recursion

27

Numeric recursion

- Can also do recursion on numbers

```c
int fact(int n) {
    if (n <= 0) {
        return 1;
    } else {
        return n * fact(n-1);
    }
}
```

ECE 590.01 (Hilton): Recursion

28

Reverse: tail recursion

```c
ll_node * reverse(ll_node * lst, ll_node * ans) {
    if (lst == NULL) {
        return ans; // Also, you can't find this scary:
    } else {
        lst = lst->next;
        ans = add_to_front(list->data, ans);
    }
}
```

ECE 590.01 (Hilton): Recursion

26

Two sides of the same coin

- Tail recursion = while loop...
  + So why not just write while loop?....
  + Some languages don't have loops
    - They don't let you change something once you make it can only make new things
  + Different ways of thinking about our invariants
    - reverse(x,y) = reverse(x) appended to y
    - May help us understand correctness better
    - May make things much easier with mutually recursive functions
  + Very complex looping structures

- Head recursion: not just a loop
  + Could be done with loop + explicit stack

- But first: numeric recursion

Numeric = Structural

- Numeric Recursion = Structural Recursion
  + Just recursing on the (abstracted away) structure of numbers
  + Mathematically, natural numbers have a structure
    - Zero
    - Or Successor of Natural Number

- Could think of them as

```c
struct nat {
    struct nat * pred;
};
```

//would be inefficient to do normal math
Numeric Recursion

- Would not actually want to implement numbers this way
  - Quite inefficient (but doable)
  - Seems like a fun exercise though (hmmm...)
- Why talk about it?
  - Good mental model for recursion
  - Master structural recursion
  - Think of the “structure” of natural numbers
  - Need to deal with negative numbers?
  - Need a different structure

Mutual Recursion

- Let’s suppose we have a simple programming language:
  - We have three types of l-values
    - An identifier (variable: x, y, z)
    - An l-value DOT an identifier (field access: x.y, z.q.a)
    - An l-value [ an expression ] (array access: a[3], a[b+c], a[y.z])

L-value = thing that can appear on left of assignment

Simple, but contrived example

- Even/odd: N is even if n-1 is odd and N is odd if N-1 is even

```c
bool isEven(int n) {
    if (n == 0) {
        return true;
    } return isOdd(n-1); }

bool isOdd(int n) {
    if (n == 0) {
        return false;
    } return isEven(n-1); }
```

- Only complexity: need prototype for forward reference
  ```c
  bool isOdd(int n);
  ```
  ```c
  bool isEven(int n) {
    if (n == 0) {
        return true;
    } return isOdd(n-1); }
  ```
Recursion on Hwk3

- Directories are naturally recursive
  - Directories contain other directories
  - Twist: can have loops
    - Due to symlinks and hardlinks
- On hwk3: will need to process directories recursively
  - Look at file name
    - Regular file? Base case
    - Directory? Recursively call your function

Head Recursion

- Head recursion is not just a simple loop
  - Could replicate with explicit stack data structure
  - But why not just use call stack?

  - Especially good for things like BSTs
    - Need to recurse left then recurse right
    - Stack keeps track of "all things to do"

  - Example:
    - Sum elements of binary tree of ints
    - Let's do in terminal