CS1101S T03D

SESSION 5

List Processing

A gift from data abstraction

Map

Apply lambda to all elements

```
function map(f, xs) {
    return is_null(xs)
           ? null
           : pair(f(head(xs)), map(f, tail(xs)));
}
```

Filter

Get list of elements where lambda return true

```
function filter(pred, xs) {
    return is_null(xs)
           ? null
           : pred(head(xs))
           ? pair(head(xs), filter(pred, tail(xs)))
           : filter(pred, tail(xs));
}
```

Accumulate

Iterate through the elements, process the current element and previous results

```
function accumulate(op, initial, xs) {
    return is_null(xs)
           ? initial
           : op(head(xs), accumulate(op, initial, tail(xs)));
}
```

List Functions

Documentations / Summary

function	**high-level idea**	**return**
map(f, xs)	Edit all elements of xs with f	list (new)
filter(pred, xs)	Get list of elements of xs where pred(elem) === true	list (new)
accumulate(op, initial, xs)	Iterate through the elements, return operation on element and previous result	return of op

```
function map(f, xs) {
    return is_null(xs)
           ? null
           : pair(f(head(xs)), map(f, tail(xs)));
}
function filter(pred, xs) {
    return is_null(xs)
           ? null
           : pred(head(xs))
           ? pair(head(xs), filter(pred, tail(xs)))
           : filter(pred, tail(xs));
}
function accumulate(op, initial, xs) {
    return is_null(xs)
           ? initial
           : op(head(xs), accumulate(op, initial, tail(xs)));
}
```

Tree

Definition?

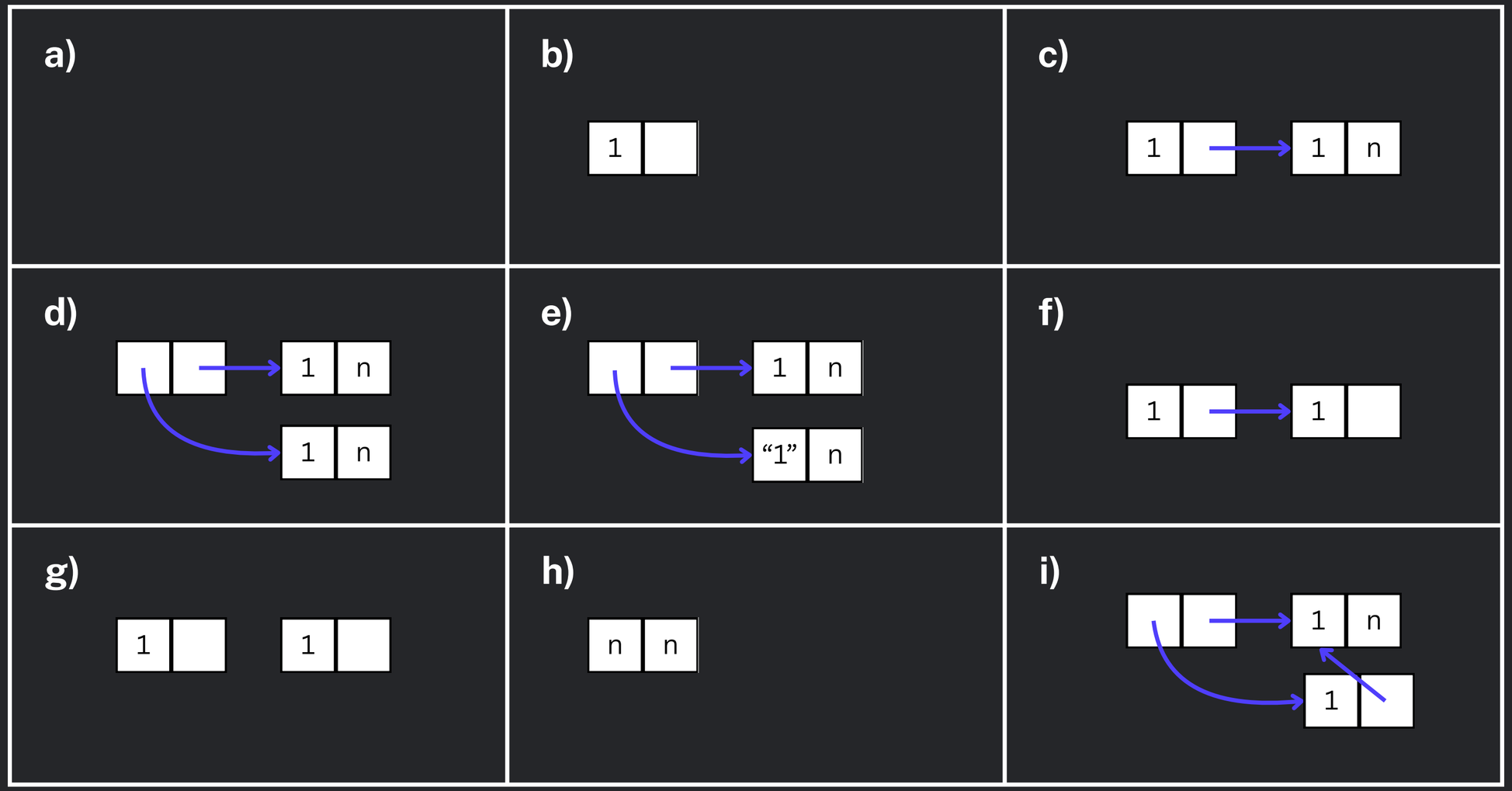
Tree

Definition?

A tree of a certain data type is:
either **null**
or a **pair**
- whose tail is a tree of that data type and
- whose head is
  - either of that data type
  - or a tree of that data type

Are these Trees?

Let n be null



Tree of <Data Type>

Why not just Tree?

Continuous Passing Style

Recursion → Iteration

```
function append(xs, ys) {  // Recursive process
    return is_null(xs)
           ? ys
           : pair(head(xs), append(tail(xs), ys));
}

function app(current_xs, ys, c) {  // Iterative process
    return is_null(current_xs)
           ? c(ys)
           : app(tail(current_xs), ys,
                 x => c(pair(head(current_xs), x)));
}
function append_iter(xs, ys) {
    return app(xs, ys, x => x);
}
```

Show
Playgr

```
function append(xs, ys) {  // Recursive process
    return is_null(xs)
           ? ys
           : pair(head(xs), append(tail(xs), ys));
}

function app(current_xs, ys, c) {  // Iterative process
    return is_null(current_xs)
           ? c(ys)
           : app(tail(current_xs), ys,
                 x => c(pair(head(current_xs), x)));
}
function append_iter(xs, ys) {
    return app(xs, ys, x => x);
}
```

Show
Playgr

CPS: store your delayed operations in a lambda :)

Pros: Technically Iterative!
Cons: Expensive final step :(

Studio Sheet

yay!

In-Class Studio Sheet

studio-S6-in-class.pdf