Week 06 Studio
List and Tree Processing, Programming Language Processing

CS1101S AY21/22 Semester 1
Studio 05E

Yan Xiaozhi (David)
@david_eom
yan_xiaozhi@u.nus.edu

13 Sep 2021

Admin

- Contact tracing (QR code + class photo)
- Mastery check 1 topics:
  - Scoping
  - Higher order function
  - Substitution model
  - Iterative / recursive process
- Preferably next week by Tuesday (this week also can)
- PM me with your preferred timing!

Avoid Magic Numbers

★★☆☆ 🏆 Can improve code quality using technique: avoid magic numbers

When the code has a number that does not explain the meaning of the number, it is called a "magic number" (as in "the number appears as if by magic"). Using a *named constant* makes the code easier to understand because the name tells us more about the meaning of the number.

📦 Example:

👎 **Bad**

```
1 return 3.14236;
2 ...
3 return 9;
```

👍 **Good**

```
1 static final double PI = 3.14236;
2 static final int MAX_SIZE = 10;
3 ...
4 return PI;
5 ...
6 return MAX_SIZE - 1;
```

Similarly, you can have 'magic' values of other data types.

👎 **Bad**

```
1 return "Error 1432"; // A magic string!
```

In general, try to avoid any magic literals.

Reading Assessment

(3) What is the result of evaluating the following Source program?

```
const x = 1;
(x => (x => (x => 2 * x)(x + 3))(3 * x + 1))(x + 4);
```

A. 2

B. 10

C. 16

D. 20

E. 38

F. 40

(4) What is the result of evaluating the following Source program?

```
function w(w, x) {
    return x <= 1 ? x : w(w, x - 1);
}

w((w, x) => 2 * x + 1, 5);
```

A. 1

B. 9

C. 19

D. 39

E. 79

F. 159

G. Error: one or more names is/are redeclared

H. Error: wrong kind of arguments(s) or wrong number of argument(s)

(5) What is the result of evaluating the following Source program?

```
function h(f, x) {
    function h(g, x) {
        return x <= 1 ? 1 : 3 * g(f, x - 1);
    }
    return x <= 1 ? 1 : 2 * f(h, x - 1);
}
h(h, 5);
```

A. 16
B. 24
C. 32
D. 36
E. 54
F. 81
G. Error: one or more names is/are redeclared
H. Error: wrong kind of arguments(s) or wrong number of argument(s)

(6) What is the sequence of values printed by the `display` function when the following program is evaluated?

```
function fun(x) {
    if (x === 0) {
        display(x);
        return 0;
    } else {
        display(fun(x - 1));
        return x;
    }
}

fun(5);
```

A. 3 2 1 0

B. 4 3 2 1 0

C. 4 3 2 1 0 0

D. 0 1 2 3

E. 0 1 2 3 4

F. 0 0 1 2 3 4

G. None of the other options is the correct answer

(10) Given that, in Source, the left operand of a binary operation is evaluated before the right operand, what is the sequence of values printed by the `display` function when the following program is evaluated?

```
function D(m, x) {
    display(m);
    return x;
}

function fun(x) {
    return x * x;
}

D( "fun",
   fun( D( "*",
           D("2", 2) * D("3", 3)
         )
      )
 );
```

A. `"2"  "*"  "3"  "fun"`

B. `"2"  "3"  "*"  "fun"`

C. `"*"  "2"  "3"  "fun"`

D. `"fun"  "2"  "*"  "3"`

E. `"fun"  "2"  "3"  "*"`

F. `"fun"  "*"  "2"  "3"`

G. None of the other options is the correct answer

(14) What kind of process does the following function give rise to for any *integer* argument $n > 0$?

```
function f(n) {
    return n <= 1 ? 1 : (x => x)(f(n - 1));
}
```

A. An iterative process

B. A recursive process

C. A process that is neither iterative nor recursive

D. A substitution process

E. An infinite process

Recap

Equality

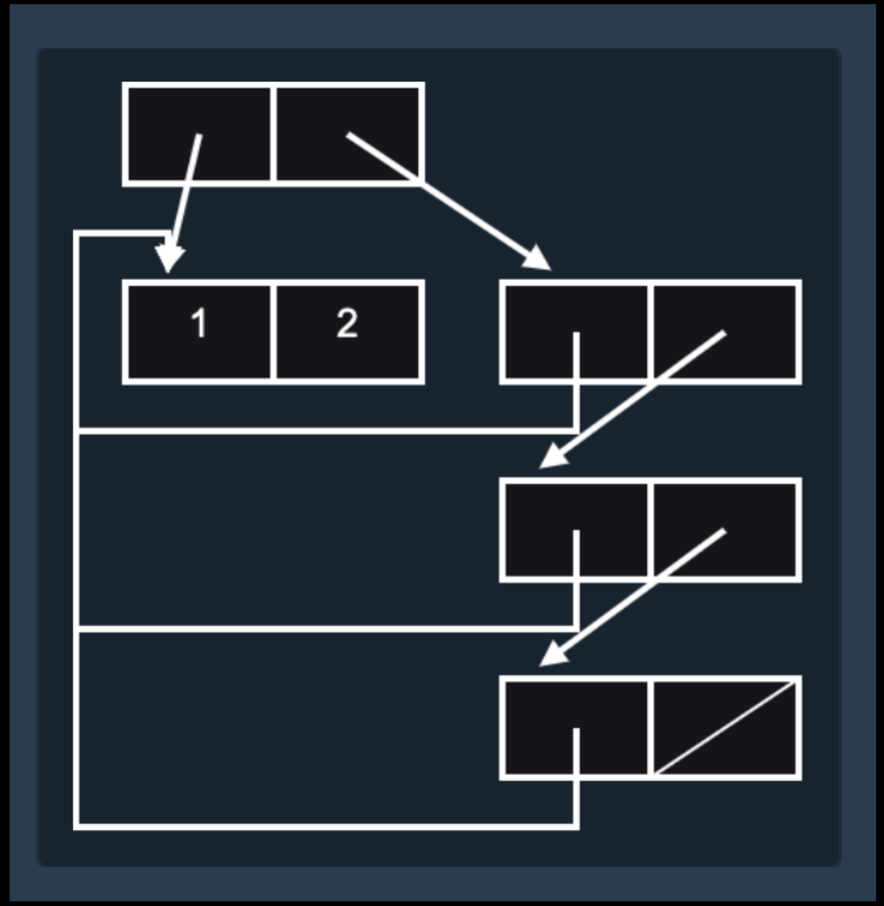
What Is It?

- Draw box and pointer diagrams for the following 3 expressions:
  - `const x = pair(1, 2);`
  - `const y = list(x, x);`
  - `const z = pair(x, pair(x, y));`

Equality

What Is It?

- What will the following expressions evaluate to?
  - `x === head(z);`
  - `x === head(tail(z));`
  - `x === head(tail(tail(z)));`
  - `x === pair(1, 2);`



Equality

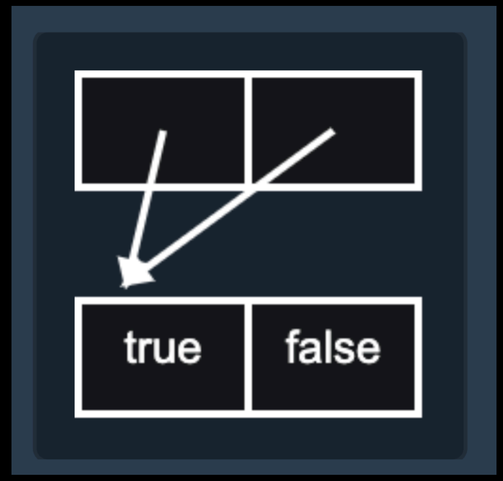
What Is It?

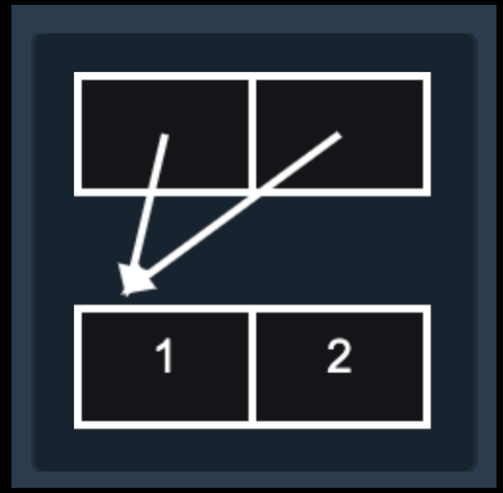
- New data structures are created upon calling pair and list
- We duplicate the value in data structures if they are of primitive types
  - Integer, boolean, float, string (only in source i think) etc.
- We use a pointer and point to the value if they are of non-primitive types
  - Pair, list, tree, etc.

Equality

What Is It?

- `const foo = pair(true, false);`
- `const bar = pair(1, 2);`
- `const baz = pair(foo, foo);`
- `const qux = pair(bar, bar);`
- `const corge = tail(baz);`
- `corge === foo;`





Equality

Equally equal vs structurally equal

- When `===` is used in source, the interpreter:
  - Looks at the value / data structure both operators are pointing to
  - Compare raw binary
  - E.g. `pair(1, 2) === pair(1, 2);`
- If we want to compare the overall structure:
  - Use `equal`

Equality

Equally equal vs structurally equal

- `equal(x, y) → {boolean}`
- Returns true if both have the same structure with respect to pair, and identical values at corresponding leave positions (places that are not themselves pairs), and false otherwise. For the "identical", the values need to have the same type, otherwise the result is false. If corresponding leaves are boolean values, these values need to be the same. If both are undefined or both are null, the result is true. Otherwise they are compared with === (using the definition of === in the respective Source language in use). Time, space: O(n), where n is the number of pairs in x.
- `equal(pair(1, 2), pair(1, 2));`

List Processing

What Have We Covered So Far?

- length
- ```
  function append(xs, ys) {
      return is_null()
          ? ys
          : pair(head(xs), append(tail(xs), ys));
  }
  ```
- ```
  function reverse(xs) {
      return is_null(xs)
          ? null
          : append(reverse(tail(xs)), list(head(xs)));
  }
  ```

List Processing

Efficient Reverse

```
• function reverse(xs) {
    function rev(original, reversed) {
      return is_null(original)
        ? reversed
        : rev(tail(original),
              pair(head(original), reversed));
    }
    return rev(xs, null);
  }
```

Higher-Order List Processing

What Is It?

- 3 pre-declared functions in Source §2
  - `map(f, xs) -> {list}`
  - `filter(pred, xs) -> {list}`
  - `accumulate(f, initial, xs) -> {value}`
- We'll skip through the implementation
- Good to internalise how they work!

Higher-Order List Processing

`map(f, xs)`

- Apply f to all the elements in `xs`
  - Original: `list(a1, a2, …, an)`
  - Mapped: `list(f(a1), f(a2), …, f(an))`
- Always return a list of the same length
- The elements in the list need not to be the same type as original
- Usage: pre-pending some item to each list, copying list (but careful!)

Higher-Order List Processing

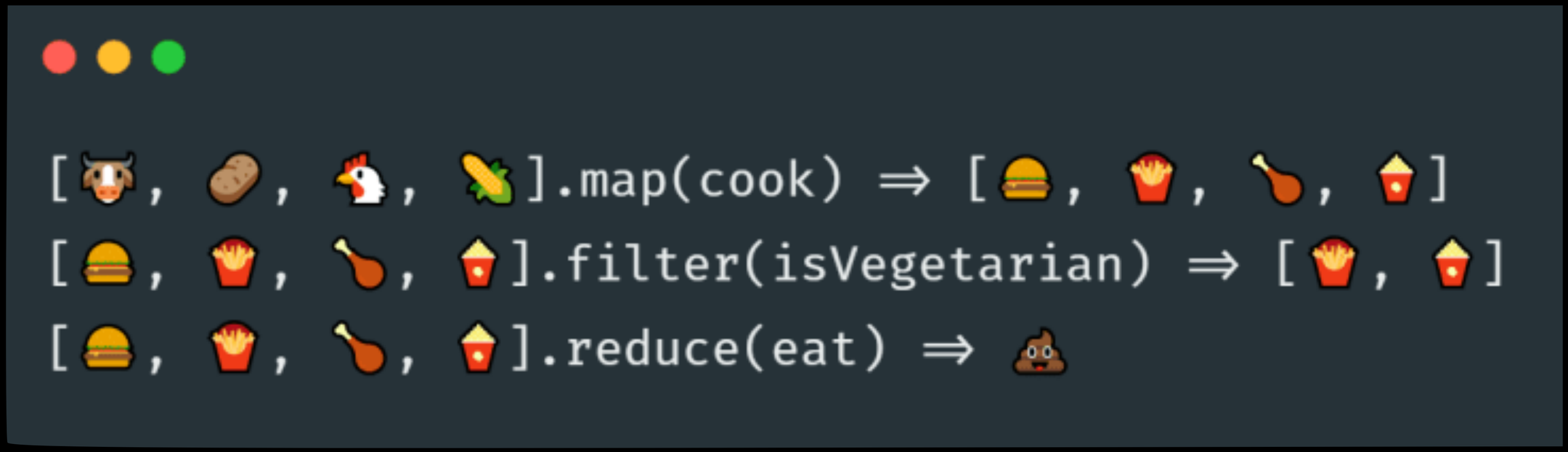
`filter(pred, xs)`

- Only keep elements in `xs` where `pred(x)` evaluates to `true`
  - Original: `list(a1, a2, …, an)`
  - Filtered: `list(a1, a4, …, am)`
- `pred` is a lambda that MUST return boolean
- Usage: filter la duh

Higher-Order List Processing

`accumulate(f, initial, xs)`

- Right-to-left folding of the elements in the list
  - Original: `list(a1, a2, …, an)`
  - Accumulated: `f(a1, f(a2, …, f(an, initial)…))`
- `f(x, y)` must be a binary function, return value must be the same as `initial`
  - x, y, and initial need not be the same type
- Usage: sum, reduction to a single result


[🐮, 🥔, 🐔, 🌽].map(cook) ⇒ [🍔, 🍟, 🍗, 🍦]
[🍔, 🍟, 🍗, 🍦].filter(isVegetarian) ⇒ [🍟, 🍦]
[🍔, 🍟, 🍗, 🍦].reduce(eat) ⇒ 💩

Tree

What Is It?

- A tree of a certain type is a list whose elements:
  - is of that type, or
  - trees of that type
- For comparison:
  - A list of a certain type is
    - `null`, or
    - a pair whose tail is a list

Tree

What Is It?

- Are the following data structures trees?
  - `list(1, 2, 3);`
  - `list(1, null, list(2, 3));`
  - `list(1, pair(2, 3));`
  - `list(1, 2, pair(3, null));`
  - `list(1, 2, “foo”, list(“bar”, “baz”));`

Tree

Higher-Order Tree Processing

- `map_tree` (analogous to `map`)
- `count_data_items` (analogous to `length`)
- Similar idea
  - Check if you have dived deep enough to reach individual elements
  - Do the mapping / counting

Tree

map_tree

```
function map_tree(f, tree) {
    return map(sub_tree =>
        !is_list(sub_tree)
            ? f(sub_tree)
            : map_tree(f, sub_tree), tree
  );
}
```

Tree

count_data_items

```
• function count_data_items(tree) {
      return is_null(tree)
          ? 0
          : (is_list(head(tree))
              ? count_data_items(head(tree))
              : 1)
            +
            count_data_items(tail(tree));
  }
```

Programming Language Processing

What Is It?

- T-Diagrams
- Interpreter
- Compilers
- Will it get tested? 🤔🤔 (idk i nvr say ah)

Any Questions?
Head over to the
most fun studio sheets imo…