CSE130 Discussion Section: Recursive data types

10.22.21

Algebraic data types

Sum type:

```
data Sum
  = C1 …
  | C2 …
  | C3 …
  ...
```

Algebraic data types

Product type:

```
data Prod
  = C F1 F2 F3

data Prod = C {
  field1 :: F1,
  field2 :: F2,
  field3 :: F3
}

field1 :: Prod -> F1
```

This is equivalent to a 3-tuple:

```
toProd :: (F1, F2, F3) -> Prod
toProd (f1, f2, f3) = C f1 f2 f3

fromProd :: Prod -> (F1, F2, F3)
fromProd (C f1 f2 f3) = (f1, f2, f3)
```

Algebraic data types

```
data Color = Red | Blue | Green

data Bool = True | False

data Foo = Color | Bool

data Bar = Bar { color :: Color, bool :: Bool }
```

How many values of type `Foo` are there?
How many values of type `Bar`?

Recursive ADTs

Product types can reference themselves!

```
data List
  = Nil
  | Cons Int List

data Tree
  = Empty
  | Node Int Tree Tree
```

This works like a linked list:

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

(`NULL` next pointer corresponds to `Nil`)