

CSE216

Foundations of Computer Science

Instructor: Zhoulai Fu

State University of New York, Korea

Many slides taken from Cornell's CS3110. Thanks!
https://www.cs.cornell.edu/courses/cs3110/2014fa/lecture_notes.php

Plan

- A review exercise
- Algebraic datatype

A review exercise on lists

- Write a function ``all_even``, of type, `int list -> bool`, which returns ``true`` if all the integers in ``lst`` are even numbers.

ALGEBRAIC DATATYPES

Recall: datatype for days

```
type day = Sun | Mon | Tue | Wed  
         | Thu | Fri | Sat
```

One-of type

Each “branch” is a *constructor*

Recall: datatype for option

- `type 'a option = None | Some of 'a`
- **None** constructor cannot carry data
- **Some** constructor can carry data

Algebraic datatypes

```
type mytype = TwoInts of int * int  
            | Str of string  
            | Pizza
```

- A constructor behaves like a function that makes values of the new type (or is a value of the new type):
 - `TwoInts : int * int -> mytype`
 - `Str : string -> mytype`
 - `Pizza : mytype`

Algebraic datatypes (2)

```
type mytype = TwoInts of int * int  
            | Str of string  
            | Pizza
```

- Any value of type **mytype** is made from *one of* the constructors
- The value contains:
 - A “tag” for “which constructor” (e.g., **TwoInts**)
 - The corresponding data (e.g., **(7, 9)**)

Accessing datatypes values with Pattern matching

```
let f (x:mytype) : int =  
  match x with  
    Pizza -> 3  
  | TwoInts(i1,i2) -> i1+i2  
  | Str s -> String.length s
```

- One branch per variant
- Each branch
 - extracts the carried data and
 - binds data to variables local to that branch

Pattern matching algebraic datatypes — summary

Syntax:

```
match e0 with
  p1 -> e1
| p2 -> e2
| ...
| pn -> en
```

For now, each *pattern* is a constructor name followed by the right number of variables (i.e., **C** or **C *x*** or **C (*x*, *y*)** or ...)

- Syntactically patterns might look like expressions
- But patterns are not expressions
 - OCaml does not evaluate patterns
 - OCaml does determine whether result of **e0** *matches* patterns

Exercise

- Define a type “point”, which is a 2 dimensional point of two floats
- Define a type “shape”, which is a point, a circle, or a rectangle. A circle is a point and a float of radius; a rectangle are two points
- Define a function `area : shape -> float`
- Define a function `center: shape -> point`

Solution

```
type point = float * float
type shape =
  | Point of point
  | Circle of point * float (* center and radius *)
  | Rect of point * point (* lower-left and upper-right corners *)

let area = function
  | Point _ -> 0.0
  | Circle (_, r) -> Float.pi *. (r ** 2.0)
  | Rect ((x1, y1), (x2, y2)) ->
    let w = x2 -. x1 in
    let h = y2 -. y1 in
    w *. h

let center = function
  | Point p -> p
  | Circle (p, _) -> p
  | Rect ((x1, y1), (x2, y2)) -> ((x2 +. x1) /. 2.0, (y2 +. y1) /. 2.0)

type point = float * float
type shape = Point of point | Circle of point * float | Rect of point * point
val area : shape -> float = <fun>
val center : shape -> point = <fun>
```

Lab exercises for syntax

1. **Define a type named point.** The type will represent a point in two-dimensional space with two integer coordinates. With this type definition, you can create 2D points like this: **Point2D (3, 4)**.
2. **Double a Point's Coordinates.** Given a point represented as Point2D (x, y), write a function to double its x and y coordinates. The result should be a new point with the coordinates (2 * x, 2 * y).
 - Input: Point2D (3, 4)
 - Output: Point2D (6, 8)
3. **Adding Two Points.** Given two points represented as Point2D (x1, y1) and Point2D (x2, y2), write a function to add their x and y coordinates respectively. The result should be a new point with the coordinates (x1 + x2, y1 + y2).
 - Input: Point2D (1, 2) and Point2D (3, 4)
 - Output: Point2D (4, 6)
4. **String Representation of a Point.** Given a point represented as Point2D (x, y), write a function to convert this point into a string representation in the format "<x, y>".
 - Input: Point2D (5, 6)
 - Output: "<5, 6>"

Lab exercises for recursion

1. Write the function **is_sorted: int list -> bool** to determine if the integers in an int list are in sorted in ascending order.
2. Write the function **insert_sorted: int -> int list -> int list**, which inserts an integer into a sorted list and preserves the sort-order. No need to check if the input is sorted.
3. Write the **insertion_sort: int list -> int list** function, using ``insert_sorted`` as a helper.