Programming Languages and Techniques (CIS120)

Lecture 4
January 22, 2016

Lists and Tuples
Which of these list expressions are equal?

a) 1 :: 2 :: []

b) (1 :: [2]) :: []

c) [1 ; 2]

d) [2 ; 1]

e) 1 :: [2]

f) [1] :: 2

1. a,c
2. a,b,c
3. a,c,e
4. a, b, c, e
5. b,c,e
6. a,c,f
7. all of them

Answer: 3
Announcements

• Please bring your clickers to class every day
  – Clicker-based attendance starts today

• Read Chapters 3 and 4 of the lecture notes
• HW#1 due Tuesday at midnight
Interactive Interlude

e-mail.ml
Tuples and Tuple Patterns
OCaml provides two ways of packaging multiple values together into a single compound value:

• **Lists:**
  – *arbitrary-length* sequence of values of a single, *fixed type*
  – example: a list of email addresses

• **Tuples:**
  – *fixed-length* sequence of values of *arbitrary types*
  – example: tuple of name, phone #, and email
Tuples

• In OCaml, tuples are created by writing the values, separated by commas, in parentheses:

```ocaml
let my_pair = (3, true)
let my_triple = ("Hello", 5, false)
let my_quaduple = (1,2,"three",false)
```

• Tuple types are written using ‘*’
  – e.g. my_triple has type:

```
string * int * bool
```
Pattern Matching Tuples

- Tuples can be inspected by pattern matching:

```ocaml
let first (x: string * int) : string =
  begin match x with
    | (left, right) -> left
  end

first ("b", 10)
⇒
"b"
```

- As with lists, the pattern follows the syntax of the values, naming the subcomponents
Mixing Tuples and Lists

• Tuples and lists can mix freely:

\[
[(1,"a"); (2,"b"); (3,"c")]
\]
: (int * string) list

\[
([1;2;3], ["a"; "b"; "c"])
\]
: (int list) * (string list)
What is the type of this expression?

\[(1, [1], [[1]])\]

1. int
2. int list
3. int list list
4. \((\text{int} \times \text{int list})\) list
5. \(\text{int} \times (\text{int list}) \times (\text{int list list})\)
6. \((\text{int} \times \text{int} \times \text{int})\) list
7. none \((\text{expression is ill typed})\)

Answer: 5