CSE216
Foundations of Computer Science

Instructor: Zhoulai Fu

State University of New York, Korea

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

- What we have learned about ocaml:

  - primitive types, functions including comparison operators, let definitions, let in expressions, if expressions, functions, types of functions, user-defined types including sum types, product types and record types, pairs, tuples, unit

- Today: Lists

Some review questions

Question #1

A **tuple** contains…

A. A fixed number of components all of which must have the same type
B. Exactly two components which may have different types
C. A fixed number of components each of which may have a different type
D. Exactly two components which must have the same type
E. I forgot to study tuples

Question #2

To access the first component of a pair, I can use…

A. The `fst` projection function

B. Pattern matching with a `let` expression

C. The `unit` expression

D. A and B

E. A and C

Question #3

What is the type of this expression?

```
let (x,y) = snd("zar",("doz",42))
in (42,y)
```

A. {x:string; y:int}

B. int*int

C. string*int

D. int*string

E. string*(string*int)

- What is the type of (1,2,3)?

- What is the type of sum_triple in
  let sum_triple ((x: int), (y: int), (z: int)): int = x+y+z

- Write a function of type int->int->int->int that returns the sum of three ints

Lists

Lists

- A list can have any number of elements (unlike pairs or tuples)

- All elements in a list have the same type

Syntax

- A list of values is a value; elements separated by semi-colons:

```
[v1;v2;…;vn]
```

- The empty list is a value:

```
[]  (* :: pronounced “nil” *)
```

- Prepend an element to beginning of list:

```
e1::e2  (* :: pronounced “cons” *)
```

Type: <type_name> list

- [1,2,3]: int list
- [true]: bool list
- [(1,3),(7,8), (9,10)] : int*int list
- [([0;1],2); ([3;4],2)]: (int list * int) list
- []: ‘a list
- Caution: semi-colons in lists, commas in tuples
- Caution: All elements in a list have the same type

Accessing lists

A list is either:

- nil
- or a head "cons-ed" onto a tail

Use **pattern matching** to access list in one of those ways:

```
let empty lst =
  match lst with
    []    -> true
  | h::t -> false
```

Example list function 1

```
let rec sum_list (lst : int list) : int =
  match lst with
    []    -> 0
  | h::t -> h + sum_list(t)
```

Example list function 2

```
let rec length (lst : int list) : int =
  match lst with
     []     -> 0
   | x::xs -> 1 + length(xs)
```

Example list function 3

```
let rec append ((lst1:'a list),(lst2:'a list))
    : 'a list =
  match lst1 with
    []    -> lst2
  | h::t -> h::append(t,lst2)
(* append is available as built-in operator @ *)
```

Lists are immutable

- No way to mutate an element of a list

- Instead, build up new lists out of old, e.g., append

Exercises

Exercise 1

What is the type of `31::[10]`?

A. int

B. int list

C. int*(int list)

D. (int*int) list

E. Not well-typed

Exercise 2

```
match [“zar”;”doz”] with
   []    -> “kitteh”
 | h::t -> h
```

To what value does the above expression evaluate?

A. “zar”
B. “doz”
C. “kitteh”
D. []
E. h

Exercise 3

- let a = [2;3;4]

- We want to append 5 in the end, what can we do?

Exercise 4

- Construct a list that has the integers 1 through 3 in it. Use the square bracket notation for lists.

- Construct the same list, but do not use the square bracket notation. Instead use :: and [].

- Construct the same list again, using the @ operator.

Exercise 5

- Write a function that returns the product of all the elements in a list. The product of all the elements of an empty list is 1

Exercise 6

- Write a function that concatenates all the strings in a list. The concatenation of all the strings in an empty list is the empty string "".