CMSC330 Organization of Programming Languages **Spring 2016**

Ocaml examples

Instructor: Dr. Anwar Mamat

Disclaimer: *These notes may be distributed outside this class only with the permission of the Instructor.*

1.1 OCaml code examples

1.1.1 Calculate the average

Listing 1: dictionary

```
1  (* calculate the average of a list of integers *)
2  let grades = [80;90;70;60];;
3  let rec fold f l acc =
4  match l with
5  []->acc
6  |h::t-> f h (fold f t acc)
7  ;;
8  let sum l = fold (fun x y ->x+y) l 0;;
9  let s = sum grades;;
10 print_int s;;
11 print_string "\n";;
12
13 let avg l =
14         let s = sum l in
15         let rec length l =
16                 match l with
17                 []->0
18                 |h::t->1 + length t
19         in s/(length l)
20 ;;
21 let v = avg grades;;
22 print_int v;;
23 print_string "\n";;
```

1.1.2 Insertion Sort

Listing 2: insertion sort

```
1  let rec sort = function
2      | [] -> []
3      | x :: l -> insert x (sort l)
4    and insert elem = function
5      | [] -> [elem]
```

```
6       | x :: l -> if elem < x then elem :: x :: l
7                   else x :: insert elem l;;
```

Listing 3: insertion sort 2

```
1  let rec sort lst =
2          match lst with
3             | [] -> []
4            | x :: l -> insert x (sort l);;
5  let rec insert elem lst =  match lst with
6      | [] -> [elem]
7      | x :: l -> if elem < x then elem :: x :: l
8                  else x :: insert elem l;;
```

1.1.3 List of functions

Listing 4: apply list of functions to a list

```
1  \label{mapmap}
2  (* in this example, we will apply a list of functions to a list
3   and return the result as a list of list *)
4  let list= [2;3;6;9];;
5  let double x = x *2;;
6  let halve x = x/2;;
7  let self x = x;;
8  let square x = x * x;;
9  let flist=[double;halve;self;square];;
10 let rec map_map fl ll =
11         match fl with
12                 []->[]
13                 |h1::t1->
14                         (let rec map f l=
15                                 match l with
16                                         []->[]
17                                         |h::t->f h::map f t
18                         in map h1 ll
19                         )::map_map t1 ll
20 ;;
21
22 map_map flist list;;
```

Listing 5: Result

```
1  Result:
2  [[4; 6; 12; 18]; [1; 1; 3; 4]; [2; 3; 6; 9]; [4; 9; 36; 81]]
```

In the example in Listing **??**, if we add following two functions to the function list.

Listing 6: more functions

```
1  let is_even x = if x mod 2 = 0 then true else false;;
```

```
let to_str x = string_of_int x;;
let flist=[double;halve;self;square;is_eve;to_str];;
```

Does it work? Why?

1.1.4 Explode: String to list

Listing 7: Result

```
let explode s =
  let rec expl i l =
    if i < 0 then l else
    expl (i - 1) (s.[i] :: l) in
  expl (String.length s - 1) [];;
```

1.1.5 Implode: Character list to string

Listing 8: Result

```
let implode l =
  let result = String.create (List.length l) in
  let rec imp i = function
  | [] -> result
  | c :: l -> result.[i] <- c; imp (i + 1) l in
  imp 0 l;;
```

1.1.6 Read a File

Listing 9: Read a file

```
(* Read lines from a text file. Use regular expression to
replace the first letter of each line with A *)

let change name =
        Str.replace_first (Str.regexp "[A-Z]") "ABC" name
;;

let read_file file_name =
        let in_file = open_in file_name in
                try
                        while true do
                                let line = input_line in_file in
                                        print_endline (change line)
                        done
                with End_of_file ->
                                close_in in_file

;;
```

```
19
20
21 read_file "names.txt";;
```

1.1.7 OCaml code examples

Listing 10: is vowel

```
1 let is_vowel c=
2         c='a' || c='e' || c='o' || c ='i' || c = 'u'
3 ;;
```

Listing 11: is vowel

```
1 let is_vowel c=
2         c='a' || c='e' || c='o' || c ='i' || c = 'u'
3 ;;
```

Listing 12: is vowel

```
1 let is_vowel c=
2         match c with
3                 |'a'->true
4                 |'o'->true
5                 |'u'->'e'
6                 |'e'->true
7                 |->'i'->true
8 ;;
```

Listing 13: is vowel

```
1
2 let is_vowel c =
3 match c with
4 'a'|'o'|'u'|'e'|'i'->true
5 |_->false
6 ;;
```

Listing 14: isnil

```
1
2 let isnil list =
3         match list with
4                 |[]->true
5                 |_->false
6 ;;
```

Listing 15: length of a list

```
1
2 let rec length list=
```

```
3   match list with
4     []->0
5     |h::t-> 1 + length t
6 ;;
```

Listing 16: reverse a list

```
1
2 let rec rev list =
3   match list with
4     []->[]
5     |h::t->rev t @[h]
6 ;;
```

Listing 17: sum of a list of integers

```
1 let rec sum list=
2   match list with
3     []->0
4     |h::t-> h + sum t
5 ;;
```

Listing 18: Append a list to another list

```
1 let rec append a b =
2   match a with
3     []->b
4     |h::t->h::append t b
5 ;;
```

Listing 19: A list of integer in a given range

```
1 let rec range a b =
2   if a > b then []
3   else a::range (a+1) b;;
```

Listing 20: range 5 10

```
1   let r = range 5 10;;
```

Listing 21: first integer of the list

```
1 let first l =
2   match l with
3     |[]->0
4     |h::_ ->h;;
```

Listing 22: last integer of the list

```
1 let rec last l =
2   match l with
3     |[] -> 0
```

```
4       |[x]->x
5       |h::t->last t
6   ;;
```

Listing 23: factorial

```
1   let rec fact n =
2     if n = 0 then 1
3     else n * fact (n-1);;
```

Listing 24: concat a list

```
1   let rec concat l =
2     match l with
3       |[]->""
4       |h::t->h ^ concat t;;
```

Listing 25: map

```
1   let rec map f l =
2     match l with
3       []->[]
4       |h::t-> f h::(map f t)
5   ;;
```

Listing 26: fold

```
1   let rec fold (f,a,l) =
2     match l with
3       []->a |(h::t)->fold (f,f(a,h),t);;
4
5   let next (a,_)=a+1;;
6   fold (next, 0, [1;2;3;4;6]);;
```

Listing 27: reverse a list using fold

```
1   let prepend(a,x) = x::a;;
2   fold (prepend, [], [1;2;3;4;5;6;7]);;
```

Listing 28: sum of a list

```
1   let sum list=
2           fold ((fun(a,x)->a+x),0,list)
```

Listing 29: sum of a list

```
1   let sum list=
2           let add (a,x)=a+x in
3                   fold (add,0,list)
4   ;;
```

Listing 30: merge 2 lists

```
let rec merge l1 l2 =
  match l1 with
    []->l2
    |a::t-> h::merge l2 t;;
```

Listing 31: insert an item to a sorted list

```
let rec insert x l =
  match l with
    []->[x]
    |h::t->if x < h then x::h::t
       else h::insert x t;;
```

Listing 32: insertion sort

```
let rec sort l=
  match l with
    |[]->[];
    |h::t-> print_int h;insert h (sort t)
;;
```

1.1.8 Number to Word

Listing 33: Number to Word Conversion

```
(*
        This program converts a number to the english word
        15 => fifteen
        123 => one hundred twenty three
*)

let get_ones x =
        match x with
                |0->""
                |1 -> "one"
                |2 -> "two"
                |3 ->"three"
                |4->"four"
                |5->"five"
                |6->"six"
                |7->"seven"
                |8->"eight"
                |9->"nine"
                |10 ->"ten"
                |11 ->"eleven"
                |12 ->"twelve"
                |13->"thirteen"
                |14 ->"fourteen"
                |15 ->"fifteen"
```

```
25                  |16 ->"sixteen"
26                  |17 ->"seventeen"
27                  |18 ->"eighteen"
28                  |19 ->"nineteen"
29                  |_->""
30  ;;
31
32  let get_tens x =
33          match x with
34          |2 ->"twenty"
35          |3->"thirty"
36          |4->"forty"
37          |5->"fifty"
38          |6->"sixty"
39          |7->"seventy"
40          |8->"eighty"
41          |9->"ninety"
42          |_->""
43  ;;
44
45
46  let rec convert num =
47          let aux (d, str)=
48                  let t1 = num / d in
49                  let t2 = num mod d in
50                  (convert t1) ^ str ^ (convert t2) in
51          if num >= 1000000000 then
52                  aux (1000000000, "␣billion␣")
53          else if num >= 1000000 then
54                  aux (1000000, "␣million␣")
55          else if num >= 1000 then
56                  aux (1000, "␣thousand␣")
57          else if num >= 100 then
58                  aux (100, "␣hundred␣")
59          else if num >= 20 then
60                  let t1 = num / 10 in
61                  let t2 = num mod 10 in
62                          (get_tens t1) ^ "␣" ^ (convert t2)
63          else
64                  get_ones num
65  ;;
66
67
68  let n = 30;;
69  print_int n;;
70  print_newline();;
71  print_string (convert n);;
72  print_newline();;
```

References

[OCaml from the very beginning] John Whittington *Coherent Press*