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

```ocaml
(* calculate the average of a list of integers *)
let grades = [80;90;70;60];;
let rec fold f l acc =
match l with
  [] -> acc
| h::t -> f h (fold f t acc)

let sum l = fold (fun x y -> x+y) l 0;;
let s = sum grades;;
print_int s;;
print_string newline;;

let avg l =
  let s = sum l in
  let rec length l =
    match l with
      [] -> 0
    | h::t -> 1 + length t
  in
  s / (length l)

let v = avg grades;;
print_int v;;
print_string newline;;
```

1.1.2 Insertion Sort

Listing 2: insertion sort

```ocaml
let rec sort = function
  | [] -> []
| x :: l -> insert x (sort l)
and insert elem = function
  | [] -> [elem]
```
Listing 3: insertion sort 2

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

```
let rec sort lst =  
  match lst with  
  | [] -> []  
  | x :: l -> insert x (sort l) ;;
let rec insert elem lst =  
  match lst with  
  | [] -> [elem]  
  | x :: l -> if elem < x then elem :: x :: l  
  else x :: insert elem l ;;
```

1.1.3 List of functions

Listing 4: apply list of functions to a list

```
\label{mapmap}
(* in this example, we will apply a list of functions to a list  
and return the result as a list of list *)
let list= [2;3;6;9] ;;
let double x = x * 2 ;;
let halve x = x / 2 ;;
let self x = x ;;
let square x = x * x ;;
let flist=[double;halve;self;square] ;;
let rec map_map f l l l =  
  match f l with  
  [] -> []  
  | h1 :: t l l -> (let rec map f l l l =  
    match l with  
    [] -> []  
    | h :: t -> f h :: map f t  
    in map h l l l ) :: map_map t l l l ;;
map_map flist list ;;
```

Listing 5: Result

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

In the example in Listing 7, if we add following two functions to the function list.

Listing 6: more functions

```
let is_even x = if x mod 2 = 0 then true else false ;;
```
2  let to_str x = string_of_int x;;
3  let flist = [double; halve; self; square; is_eve; to_str];;;

Does it work? Why?

1.1.4 Explode: String to list

Listing 7: Result
1  let explode s =
2    let rec expl i l =
3      if i < 0 then l else
4      expl (i - 1) (s.[i] :: l) in
5    expl (String.length s - 1) [];;

1.1.5 Implode: Character list to string

Listing 8: Result
1  let implode l =
2    let rec imp i = function
3      | [] -> result
4      | c :: l -> result.[i] <- c; imp (i + 1) l in
5    imp 0 l;;

1.1.6 Read a File

Listing 9: Read a file
1  (* Read lines from a text file. Use regular expression to
2     replace the first letter of each line with A *)
3
4  let change name =
5    Str.replace_first (Str.regexp "[A-Z"]") "ABC" name
6  ;;
7  let read_file file_name =
8    let in_file = open_in file_name in
9    try
10      while true do
11        let line = input_line in_file in
12        print_endline (change line)
13      done
14    with End_of_file ->
15      close_in in_file
16  ;;
1.1.7 OCaml code examples

Listing 10: is vowel

```ocaml
let is_vowel c =
  c='a' || c='e' || c='o' || c='i' || c='u';
```

Listing 11: is vowel

```ocaml
let is_vowel c =
  c='a' || c='e' || c='o' || c='i' || c='u';
```

Listing 12: is vowel

```ocaml
let is_vowel c =
  match c with
  | 'a' -> true
  | 'o' -> true
  | 'u' -> 'e'
  | 'e' -> true
  | _ -> 'i' -> true
```

Listing 13: is vowel

```ocaml
let is_vowel c =
  match c with
  | 'a' | 'o' | 'u' | 'e' | 'i' -> true
  | _ -> false
```

Listing 14: isnil

```ocaml
let isnil list =
  match list with
  | [] -> true
  | _ -> false
```

Listing 15: length of a list

```ocaml
let rec length list =
```
match list with
   |[] -> 0
   |h::t -> 1 + length t

Listing 16: reverse a list

let rec rev list =
  match list with
  |[] -> []
  |h::t -> rev t @ [h]

Listing 17: sum of a list of integers

let rec sum list =
  match list with
  |[] -> 0
  |h::t -> h + sum t

Listing 18: Append a list to another list

let rec append a b =
  match a with
  |[] -> b
  |h::t -> h::append t b

Listing 19: A list of integer in a given range

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

Listing 20: range 5 10

let r = range 5 10;;

Listing 21: first integer of the list

let first l =
  match l with
  |[] -> 0
  |h::t -> h

Listing 22: last integer of the list

let rec last l =
  match l with
  |[] -> 0
Listing 23: factorial

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

Listing 24: concat a list

```ml
let rec concat l =
  match l with
  | [] -> ""
  | h::t->h @ concat t ; ;
```

Listing 25: map

```ml
let rec map f l =
  match l with
  | [] -> []
  | h::t-> f h : : ( map f t )
  ; ;
```

Listing 26: fold

```ml
let rec fold ( f, a, l ) =
  match l with
  | [] -> a
  | ( h::t) -> fold ( f, f ( a, h ), t ) ; ;
```

Listing 27: reverse a list using fold

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

Listing 28: sum of a list

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

Listing 29: sum of a list

```ml
let sum list=
  let add ( a, x ) = a + x in
  fold ( add, 0, list )
  ; ;
Listing 30: merge 2 lists

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

Listing 31: insert an item to a sorted list

```ml
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

```ml
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

```ml
(*
   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"
```
let get_tens x =
  match x with
  | 2 ->"twenty"  
  | 3 ->"thirty"  
  | 4 ->"forty"  
  | 5 ->"fifty"  
  | 6 ->"sixty"  
  | 7 ->"seventy"  
  | 8 ->"eighty"  
  | 9 ->"ninety"  
  | _ ->""  
|

let rec convert num =
  let aux (d, str) =
    let t1 = num / d in
    let t2 = num mod d in
    (convert t1) ^ str ^ (convert t2) in
    if num >= 1000000000 then
    aux (1000000000, "_,billion,_")  
    else if num >= 100000 then
    aux (100000, "_,million,_")  
    else if num >= 1000 then
    aux (1000, "_,thousand,_")  
    else if num >= 100 then
    aux (100, "_,hundred,_")  
    else if num >= 20 then
    let t1 = num / 10 in
    let t2 = num mod 10 in
    (get_tens t1) ^ "_" ^ (convert t2)  
    else
    get_ones num  
|

let n = 30;
print_int n;;
print_newline();
print_string (convert n);;
print_newline();
References

[OCaml from the very beginning]  JOHN WHITTINGTON *Coherent Press*