### 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 "\n";;

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 "\n";;
```

#### 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]
```

1.1.3 List of functions

Listing 4: apply list of functions to a list

```ocaml
(* 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 f list = [double; halve; self; square];;
let rec map_map fl ll =
    match fl with
    | [] -> []
    | h1::tl1 ->
        (let rec map f l =
            match l with
            | [] -> []
            | h::t -> f h::map f t
        in map h1 ll)
    :: map_map tl1 ll
;
map_map f list;;
```

Listing 5: Result

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

```ocaml
let is_even x = if x mod 2 = 0 then true else false;;
```
2 \texttt{let to\_str } x = \texttt{string\_of\_int } x ; ; \\
3 \texttt{let flist } = \texttt{[double; halve; self; square; is\_e\_\_e\_\_ve; to\_str ]; ;}

Does it work? Why?

1.1.4 Explode: String to list

Listing 7: Result

1 \texttt{let explode } s = \\
2 \hspace{1em} \texttt{let rec expl i l =} \\
3 \hspace{2em} \texttt{if } i < 0 \texttt{then } l \texttt{else} \\
4 \hspace{3em} \texttt{expl (i - 1) (s.[i] :: l) in} \\
5 \hspace{2em} \texttt{expl (String.length s - 1) [ ]; ;}

1.1.5 Implode: Character list to string

Listing 8: Result

1 \texttt{let implode l =} \\
2 \hspace{1em} \texttt{let result = String.create (List.length l) in} \\
3 \hspace{1em} \texttt{let rec imp i = function} \\
4 \hspace{2em} | [] -> result \\
5 \hspace{2em} | c :: l -> result.[i] <- c; imp (i + 1) l in} \\
6 \hspace{1em} imp 0 l ; ;

1.1.6 Read a File

Listing 9: Read a file

1 \texttt{( \ast \ Read \ lines \ from \ a \ text \ file. \ Use \ regular \ expression \ to} \\
2 \texttt{replace \ the \ first \ letter \ of \ each \ line \ with \ A \ *)} \\
3 \texttt{)} \\
4 \texttt{let change name =} \\
5 \hspace{1em} \texttt{Str.replace.first (Str.regexp "[A-Z]" ) "ABC" name} \\
6 \hspace{1em} \texttt{; ;} \\
7 \texttt{let read_file file_name =} \\
8 \hspace{1em} \texttt{let in_file = open_in file_name in} \\
9 \hspace{1em} \texttt{try} \\
10 \hspace{2em} \texttt{while true do} \\
11 \hspace{3em} \texttt{let line = input_line in_file in} \\
12 \hspace{3em} \texttt{print_endline (change line)} \\
13 \hspace{2em} \texttt{done} \\
14 \hspace{1em} \texttt{with End_of_file ->} \\
15 \hspace{2em} \texttt{close_in in_file} \\
16 \texttt{; ;}
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 is_nil 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
1. \([x] \rightarrow x\)
2. \([h::t] \rightarrow \text{last } t\)

Listing 23: factorial

1. \textbf{let rec} \textbf{fact} \textbf{n} =
2. \textbf{if} \textbf{n} = 0 \textbf{then} 1
3. \textbf{else} \textbf{n} * \textbf{fact} (\textbf{n}-1);

Listing 24: concat a list

1. \textbf{let rec} \textbf{concat} \textbf{l} =
2. \textbf{match} \textbf{l} with
3. \textbf{|[]} \rightarrow ""
4. \textbf{|h::t} \rightarrow \textbf{h} \hat{\text{ }} \textbf{concat} \textbf{t};

Listing 25: map

1. \textbf{let rec} \textbf{map} \textbf{f l} =
2. \textbf{match} \textbf{l} with
3. \textbf{|[]} \rightarrow \textbf{|[]}
4. \textbf{|h::t} \rightarrow \textbf{f h} : : (\textbf{map} \textbf{f} \textbf{t})
5. ;;

Listing 26: fold

1. \textbf{let rec} \textbf{fold} (\textbf{f}, \textbf{a}, \textbf{l}) =
2. \textbf{match} \textbf{l} with
3. \textbf{|[]} \rightarrow \textbf{a} \mid (\textbf{h::t}) \rightarrow \textbf{fold} (\textbf{f}, \textbf{f(a,h)}, \textbf{t});
4. \textbf{let} \textbf{next} (\textbf{a}, \_)=\textbf{a} + 1;
5. \textbf{fold} (\textbf{next}, 0, \textbf{[1;2;3;4;6]});

Listing 27: reverse a list using fold

1. \textbf{let prepend}\textbf{(a,x)} = \textbf{x} : : \textbf{a};
2. \textbf{fold} (\textbf{prepend}, \textbf{[]}, \textbf{[1;2;3;4;5;6;7]});

Listing 28: sum of a list

1. \textbf{let sum list}=
2. \textbf{fold} ((\textbf{fun} (\textbf{a,x}) \rightarrow \textbf{a+x}), 0, \textbf{list})

Listing 29: sum of a list

1. \textbf{let sum list}=
2. \textbf{let add (a,x)=a+x in}
3. \textbf{fold} (\textbf{add}, 0, \textbf{list})
4. ;;
Listing 30: merge 2 lists

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

Listing 31: insert an item to a sorted list

```ocaml
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

```ocaml
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

```ocaml
(@

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