OCaml PRO Training – Initiation to OCaml for Imperative Programmers
Bloomberg London - March 2015

Intermediate Level Practice Sheet *,**

Topics

- Algebraic data types
- Simple modules
- Advanced I/Os

Exercise 1 – Generic number type

Question 1.1 – Define a type `number` that can contain an `int` or a `float`.

Question 1.2 – Define the `number_is_zero` predicate.

Question 1.3 – Define `int_of_number` and `float_of_number`.

Question 1.4 – Define `(+)`, `(-)`, `(*)` and `(/)` over numbers, that should always take a sensible representation

Question 1.5 – Put these definitions in a module, and make the definition abstract using an interface file.

Exercise 2 – Lists redefined

Question 2.1 – Write a type `nlist` that allows to get the length of a list in constant time. Define it using your own list structure, not wrapping OCaml's lists.

Question 2.2 – Implement `nlist_length` (equivalent to `List.length`), `nlist_cons` (equivalent to `(::)`)and `nlist_append` (equivalent to `(@)`).

Question 2.3 – Make the type nlist private using a signature. Explain whey it is better thant making it abstract.

Exercise 3 – Binary search trees

Question 3.1 – A binary tree is a tree whose nodes have exactly two children and bear a value. Define a type `'a bin` of binary trees whose nodes hold values of type `'a`. The empty tree must de representable.

Question 3.2 – Write a `to_list` function that converts a binary tree to a list.

Question 3.3 – Define an `iter` function, that iterates over a tree in infix order.

Question 3.4 – The tree is called a binary search tree if, all the children on the left of a node hold lower values than itself, and all its right children hold greater values. Write the `valid_search_tree` predicate.

Question 3.5 – Write an insert function that builds a new tree with an additional value at the right place, so that if the original tree was a binary search tree, the result is so too.

Question 3.6 – Isolate the type of trees in a modules, and make it abstract with an interface. Define appropriate constructor functions so that only binary search trees can be cosntructed outside of the module.

Question 3.7 – Using functions from this module, functions write a `list_sort_unique` function that takes a list, and returns a sorted version without duplicates.

Exercise 4 – Advanced I/Os

Question 4.1 – Define a type `'a vfs` to represent a virtual file system structure in memory. You must handle directories, files and their contents as generic values.

Question 4.2 – Write a function `print_vfs` that prints the structure as in the following example, without displaying the contents for now. You can do the identation by hand or let options from the `Format` module do it for you.

```
/dir = [
  /dir_a = [
    /file_z.asc
    /file_w.asc
  ]
  /dir_b = []
  /dir_c = [
    /file_x.txt
    /file_y.txt
  ]
]
```

Question 4.3 – Assuming that the virtual file contents are lists of characters, display them wrapped at column 80 as in the following example. Here again, the `Format` module can do it automatically.

```
/dir = [
  /dir_a = [
    /file_z.asc = "hello world"
    /file_w.asc =
      "hello world hello world hello world hello world hello world hello wo
       rld hello world hello world hello world hello world hello world hell
       o world"
  ]
]
```

Question 4.4 – Using the `Sys` module, write a function `read_dir` that reads a directory recursively and builds its OCaml representation. Try to read and pretty print an example directory..

Intermediate Level Practice Sheet

OLUTIONS – SOLUTIONS – SOLUTIONS – SOLUTION

Solution to question 1.1

```
type number = Int of int | Float of float
```

Solution to question 1.2

```
let number_is_zero = function
  | Int 0 | Float 0. -> true
  | _ -> false
```

Solution to question 1.3

```
let int_of_number = function
  | Int i -> i
  | Float f -> int_of_float f
let float_of_number = function
  | Float f -> f
  | Int i -> float_of_int i
```

Solution to question 1.4

```
let op float_op int_op x y =
  let float_or_int f =
    let i = int_of_float f in
    if float_of_int i = f then Int i else Float f in
  match (x, y) with
  | Int x, Int y -> Int (int_op x y)
  | Int x, Float y -> float_or_int (float_op (float_of_int x) y)
  | Float x, Int y -> float_or_int (float_op x (float_of_int y))
  | Float x, Float y -> float_or_int (float_op x y)
let ( + ) = op ( +. ) ( + )
let ( - ) = op ( -. ) ( - )
let ( * ) = op ( *. ) ( * )
let ( / ) = op ( /. ) ( / )
```

Solution to question 1.5

```
type number
val number_is_zero ; number -> bool
val int_of_number : number -> int
val float_of_number : number -> float
val ( + ) : number -> number -> number
val ( - ) : number -> number -> number
val ( * ) : number -> number -> number
val ( / ) : number -> number -> number
```

Solution to question 2.1

```ocaml
type 'a nlist = Nnil | Ncons of int * 'a * 'a nlist
```

Solution to question 2.2

```ocaml
let nlist_length = function
  | Nnil -> 0
  | Ncons (n, _, _) -> n
let nlist_cons x l = Ncons (nlist_length l + 1, x, l)
let rec nlist_append l1 l2 = match l1 with
  | Nnil -> l2
  | Ncons (_, x, t2) ->
      nlist_add x (nlist_append t2 l2)
```

Solution to question 2.3

```ocaml
type 'a nlist = private Nnil | Ncons of int * 'a * 'a nlist
val nlist_length : 'a nlist -> int
val nlist_cons : 'a -> 'a nlist -> 'a nlist
val nlist_append : 'a nlist -> 'a nlist -> 'a nlist
```

Solution to question 3.1

```ocaml
type 'a bin = Node of 'a bin * 'a * 'a bin | Empty
```

Solution to question 3.2

```ocaml
let rec to_list = function
  | Node (l,x,r) -> to_list l @ x :: to_list r
  | Empty -> []
```

Solution to question 3.3

```ocaml
let rec iter f = function
  | Node (l,x,r) -> iter f l ; f x ; iter f r
  | Empty -> ()
```

Solution to question 3.4

```ocaml
exception Not_well_formed
let valid_search_tree =
  let rec minmax = function
    | Node (Empty, x, r) ->
      let (rmin, rmax) = minmax r in
      if x < rmin then (x, rmax) else raise Not_well_formed
    | Node (l, x, Empty) ->
      let (lmin, lmax) = minmax l in
      if x > lmax then (lmin, x) else raise Not_well_formed
    | Node (l, x, r) ->
      let (lmin, lmax) = minmax l in
      let (rmin, rmax) = minmax r in
      if x < rmin && x > lmax then (lmin, rmax) else raise Not_well_formed
    | Empty -> assert false in
  function
    | Empty -> true
    | t -> try ignore (minmax t) ; true with Not_well_formed -> false
```

Solution to question 3.5

```
let rec insert x = function
  | Empty -> Node (Empty, x, Empty)
  | Node (l, y, r) when x < y -> Node (insert x l, y, r)
  | Node (l, y, r) when x > y -> Node (l, y, insert x r)
  | n -> n
```

Solution to question 3.6

File `bin.mli`:

```
type 'a bin
val empty : 'a bin
val insert : 'a -> 'a bin -> 'a bin
val to_list : 'a bin -> 'a list
val iter : ('a -> unit) -> 'a bin -> unit
```

Solution to question 3.7

```
let list_sort_unique l =
  Bin.(to_list (List.fold_right insert l empty)) ;;
```

Solution to question 4.1

```
type 'a vfs =
  'a inode array
and 'a inode =
  | File of string * 'a
  | Dir of string * 'a inode array
```

Solution to question 4.2

```
let rec print_item = function
  | File (n, _) ->
    Format.printf "@,/%s" n
  | Dir (n, [||]) ->
    Format.printf "@,/%s␣=␣[]" n ;
  | Dir (n, items) ->
    Format.printf "@,@[<v␣2>/%s␣=␣[" n ;
    Array.iter print_item items ;
    Format.printf "@]@,]" ;
and print_vfs items =
    Format.printf "@[<v␣0>" ;
    Array.iter print_item items ;
    Format.printf "@]"
```

Solution to question 4.4

```
let rec read_dir path =
  Array.map
    (fun name ->
       let path = Filename.concat path name in
       if Sys.is_directory path then
         Dir (name, read_dir path)
       else
         File (name, read_file path))
    (Sys.readdir path)
and read_file path =
```

```
  let rec chars acc fp =
    match input_char fp with
    | c -> chars (c :: acc) fp
    | exception End_of_file -> List.rev acc in
  let fp = open_in path in
  let res = chars [] fp in
  close_in fp ; res
let () = print_vfs (read_dir Sys.argv.(1)) ;;
```