Principles of Programming, Fall 2009
Practice 8
OCaml Basic Module System and Standard Library

Woosuk Lee, Suwon Jang, Sungkeun Cho
Programming Research Lab.@SNU

November 2, 2009

1. In OCaml, Module is a set of declarations. To learn more about module system, our first step is implementing simplest module which is called '`MyQueue`'. We will show you an example module '`MyStack`'. Write your own module '`MyQueue`'.

```
module MyStack =
  struct
    type t = int list
    exception Empty
    let empty = []
    let push x t = x :: t
    let pop t =
      match t with
        [] -> raise Empty
      | h::t -> (h, t)
    let first t =
      match t with
        [] -> raise Empty
      | h::t -> h
  end
```

2. OCaml has an information hiding feature, signature. Signatures are interfaces for structures. You can define '`module type`' as an interface to module structure. Write signiture '`MYQUEUE`' to hide internal implementation of '`MyQueue`'. Sample signiture for '`MyStack`' is provided.

```
module type MYSTACK =
  sig
```

```
    type t
    exception Empty
    val empty : t
    val push : int -> t -> t
    val pop : t -> int * t
  end
```

And, confirm that restricting the Queue structure by this signature results in another view of the Queue structure where the `first` function is not accessible and the actual representation of queues is hidden. An example for `MyStack` is given as follow.

```
# module AbstractStack = (MyStack : MYSTACK);;
module AbstractStack : MYSTACK
# AbstractStack.first [1;2;3] ;;
Unbound value AbstractStack.first
```

3. Phone Book.

   A phone book contains names of people and phone number set corresponding to each person. You can usually add a pair of name and phone number into your phone book, delete informations, and lookup the specific information which you need. Implement a module `PhoneBook` of which signature can be defined as follow. Actually, this exercise is for getting used to Ocaml standard library. You'd better make use of some standard library : { `Set`, `Map`, `String`, `List`}

```
module type PHONEBOOK =
  sig
    type t
type name = string
type number = string
    exception Empty
    val empty : t
    val add : name * number -> t -> t
    val del : name * number -> t -> t
val lookup : name -> t -> number list
val pprint : t -> unit
  end
```

   An usage example of your `PhoneBook` can be represented as follow.

```
# let _ =
 let book = PhoneBook.add ("kim", "4732938") (PhoneBook.empty) in
 let book = PhoneBook.add ("kim", "4128947") book in
 let book = PhoneBook.add ("lee", "4524438") book in
 let book = PhoneBook.add ("lee", "4137438") book in
 let book = PhoneBook.del ("lee", "4137438") book in
  PhoneBook.pprint book
```

```
kim ::
    412-8947
    473-2938
lee ::
    452-4438
```