

```

1 type zero = private Dummy1
2 type _ succ = private Dummy2
3 type nil = private Dummy3
4 type 'a list = Nil | Cons of 'a * 'a list
5
6
7 (* Exercice 1 *)
8 module Exo1 =
9   struct
10     type ('a, 'n) nlist = Nil : ('a, zero) nlist | Cons : 'a * ('a, 'n) nlist
11     -> ('a, 'n succ) nlist
12     let rec map : type n. ('a -> 'b) -> ('a, n) nlist -> ('b, n) nlist = fun
13     l ->
14       match l with
15       | Nil -> Nil
16       | Cons (t, q) -> Cons (f t, map f q)
17     let rec snoc : type n. ('a, n) nlist -> 'a -> ('a, n succ) nlist = fun l
18     e ->
19       match l with
20       | Nil -> Cons (e, Nil)
21       | Cons (t, q) -> Cons (t, snoc q e)
22     let rec rev : type n. ('a, n) nlist -> ('a, n) nlist = fun l ->
23       match l with
24       | Nil -> Nil
25       | Cons (t, q) -> snoc (rev q) t
26     end
27
28 (* Exercice 2 *)
29 module Exo2 =
30   struct
31     open Exo1
32     let rec insert : type n. 'a -> ('a, n) nlist -> ('a, n succ) nlist =
33     fun x -> function
34       | Nil -> Cons(x, Nil)
35       | Cons (t, q) as l -> if t < x then Cons (t, insert x q) else Cons (x,
36     l);;
37     let rec insertion_sort : type n. ('a, n) nlist -> ('a, n) nlist =
38     function
39       | Nil -> Nil
40       | Cons (t, q) -> insert t (insertion_sort q);;
41     end
42
43 (* Exercice 3 *)
44 module Exo3 =
45   struct
46     type _ hlist = Nil : nil hlist | Cons : 't * 'q hlist -> ('t * 'q) hlist
47     let tail : type t q. (t * q) hlist -> q hlist = function (Cons (_, q)) ->
48     q
49     let add : type q. (int * (int * q)) hlist -> (int * q) hlist = function
50     | Cons (i1, Cons (i2, q)) -> Cons (i1+i2, q)
51     end
52
53 (* Exercice 4 *)
54 module Exo4 =
55   struct
56     type 't expr =

```

```

55 | Entier : int -> int expr
56 | Booleen : bool -> bool expr
57 | Plus : int expr * int expr -> int expr
58 | Egal : 't expr * 't expr -> bool expr
59
60 let rec eval : type t. t expr -> t =
61   function
62   | Entier i -> i
63   | Booleen b -> b
64   | Plus (e1, e2) -> eval e1 + eval e2
65   | Egal (e1, e2) -> eval e1 = eval e2
66 end
67
68 (* Exercice 5 *)
69 module Exo5 =
70   struct
71     open Exo4
72     type valeur = Int of int | Bool of bool
73     type code = PushI of int | PushB of bool | Add | Equ | Seq of code * code
74
75     let rec compile : type t. t expr -> code =
76     function
77     | Entier i -> PushI i
78     | Booleen b -> PushB b
79     | Plus (e1, e2) -> Seq (compile e1, Seq (compile e2, Add))
80     | Egal (e1, e2) -> Seq (compile e1, Seq (compile e2, Equ))
81
82     let rec exec code pile =
83     match code, pile with
84     | PushI i, _ -> (Int
85     i)::pile
86     | PushB b, _ -> (Bool
87     b)::pile
88     | Add, (Int i1)::(Int i2)::reste -> (Int
89     (i1+i2))::reste
90     | Equ, (Int i1)::(Int i2)::reste -> (Bool
91     (i1=i2))::reste
92     | Equ, (Bool b1)::(Bool b2)::reste -> (Bool
93     (b1=b2))::reste
94     | Seq (c1, c2), _ -> exec c2
95     | _, _ -> failwith "erreur"
96   end
97
98 (* Exercice 6 *)
99 module Exo6 =
100   struct
101     (* Attention au parenthésage des produits de types !! *)
102     (* 'a * 'b * 'c /= 'a * ('b * 'c) /= ('a * 'b) * 'c *)
103     (* Ici, il faut utiliser 'a * ('b * 'c) pour exprimer la structure de
104     pile *)
105     type ('pile1, 'pile2) code =
106     | PushI : int -> ('pile, int * 'pile) code
107     | PushB : bool -> ('pile, bool * 'pile) code
108     | Add : (int * (int * 'reste), int * 'reste) code
109     | Equ : ('a * ('a * 'reste), bool * 'reste) code

```

```

106 | Seq : ('pile1, 'pile2) code * ('pile2, 'pile3) code -> ('pile1,
'pile3) code
107 end
108
109
110 (* Exercice 7 *)
111 module Exo7 =
112 struct
113   open Exo3
114   open Exo4
115   open Exo6
116
117   let rec compile : type t stackin. t expr -> (stackin, t * stackin) code =
118     function
119     | Entier i      -> PushI i
120     | Booleen b    -> PushB b
121     | Plus (e1, e2) -> Seq (Seq (compile e1, compile e2), Add)
122     | Egal (e1, e2) -> Seq (Seq (compile e1, compile e2), Equ)
123
124   let rec exec : type pile1 pile2. (pile1, pile2) code -> pile1 hlist ->
pile2 hlist = function
125   | PushI i -> (fun pile -> Cons (i, pile))
126   | PushB b -> (fun pile -> Cons (b, pile))
127   | Add     -> (function (Cons (i1, Cons (i2, reste))) -> Cons (i1
+ i2, reste))
128   | Equ     -> (function (Cons (v1, Cons (v2, reste))) -> Cons (v1
= v2, reste))
129   | Seq (c1, c2) -> (function pile -> exec c2
(exec c1 pile))
130 end
131
132 (* Exercice 7 avec variation sur les problèmes de polymorphisme *)
133 module Exo7bis =
134 struct
135   open Exo3
136   open Exo4
137   open Exo6
138
139   type 't gcode = { contents : 'stackin. ('stackin, 't * 'stackin) code }
140
141   let pushI i = { contents = PushI i }
142   let pushB b = { contents = PushB b }
143   let add c1 c2 = { contents = Seq (c1.contents, Seq (c2.contents, Add)) }
144   let equ c1 c2 = { contents = Seq (c1.contents, Seq (c2.contents, Equ)) }
145
146   let rec gcompile : type t. t expr -> t gcode =
147     function
148     | Entier i      -> pushI i
149     | Booleen b    -> pushB b
150     | Plus (e1, e2) -> add (gcompile e1) (gcompile e2)
151     | Egal (e1, e2) -> equ (gcompile e1) (gcompile e2)
152
153   let rec exec : type pile1 pile2. (pile1, pile2) code -> pile1 hlist ->
pile2 hlist = function
154   | PushI i -> (fun pile -> Cons (i, pile))
155   | PushB b -> (fun pile -> Cons (b, pile))
156   | Add     -> (function (Cons (i1, Cons (i2, reste))) -> Cons (i1
+ i2, reste))
157   | Equ     -> (function (Cons (v1, Cons (v2, reste))) -> Cons (v1
= v2, reste))

```

```

158 | Seq (c1, c2) -> (function pile -> exec c2
(exec c1 pile))
159
160 end
161

```