UNIVERSITY OF EDINBURGH

FACULTY OF SCIENCE AND ENGINEERING

M.Sc./DIPLOMA IN ARTIFICIAL INTELLIGENCE
M.Sc./DIPLOMA IN COGNITIVE SCIENCE
M.Sc./DIPLOMA IN COMPUTER SCIENCE
M.Sc./DIPLOMA IN INFORMATICS

FUNCTIONAL PROGRAMMING AND SPECIFICATION

Friday 26 April 2002

14:00 to 15:30

Examiners:
N. Chater (External)
N. Jennings (External)
E. Hancock (External)
K. Hawick (External)
R.N. Ibbett (Convener)
J. Oberlander (Co-Convener)

INSTRUCTIONS TO CANDIDATES

Answer any TWO questions.

All questions carry equal weight.

1. Recall the definition of `'a option`:

```
datatype 'a option = NONE | SOME of 'a
```

Consider the following functions:

```
fun mapPartial f [] = []
  | mapPartial f (h::t) = case f h of NONE => mapPartial f t
                                     | SOME v => v::(mapPartial f t)

fun take(_,0) = []
  | take([],_) = raise Subscript
  | take(h::t,n) = h::take(t,n-1)
```

(a) Sometimes these functions are composed as follows:

```
take(mapPartial f s, n)
```

which can be inefficient since list elements are computed by `mapPartial` only to be discarded by `take`. In such circumstances it would be more efficient to use a function `takeMapPartial` with the type

```
('a -> 'b option) -> 'a list -> int -> 'b list
```

Implement `takeMapPartial`. [*10 marks*]

(b) Give an example of `f`, `s` and `n` for which `take(mapPartial f s, n)` and `takeMapPartial f s n` would differ. [*5 marks*]

(c) Some SML functions raise exceptions instead of explicitly indicating failure with the value `NONE` of the `option` datatype. Consider now `robustMap`, a robust version of the `map` function which traps these exceptions and continues to process the remainder of the list, with the same type as `map` itself:

```
('a -> 'b) -> 'a list -> 'b list
```

Provide *two* implementations of `robustMap`, one which uses `mapPartial` and one which does not. [*10 marks*]

2. A *multiqueue* is a collection of queues with a service discipline where the customer at the front of the longest queue is served first. A new customer might choose to join any of the existing queues by number or start another one (by joining queue $n+1$ out of n, say). One can see how many customers are in each queue; there are zero customers in queue $n+1$ of n. Attempting either to identify or to serve the front customer when there are no customers are both exceptional cases. The following SML signature describes multiqueues:

```
signature MULTIQUEUE =
  sig
    type multiqueue
    type customer
    val empty: multiqueue
    val join: multiqueue * int * customer -> multiqueue
    val customers: multiqueue * int -> int
    exception Front
    exception Serve
    val front: multiqueue -> customer
    val serve: multiqueue -> multiqueue
  end
```

Implement an SML functor which produces a multiqueue structure for a given customer type. (One way of representing a multiqueue is as a list of lists. If you choose this representation then you may use functions such as `length`, `nth`, `hd` and `tl` if you require them.) *[25 marks]*

3. Consider the following SML signature for a simple pattern matcher:

```
signature MATCH =
  sig
    type pattern = string list
    val matches : pattern * int list -> bool
  end
```

A pattern is a list of variables, represented as strings. When matching a list of variables against a list of integers, variables must be matched consistently. So, for instance, `["X","Y","X"]` matches `[1,1,1]` and `[2,1,2]` but not `[1,2,2]`.

(a) Turn the above signature into an EML specification of the problem. [*8 marks*]

(b) Give an SML structure that matches your EML signature. [*8 marks*]

(c) Explain why your structure satisfies your specification. You need not give a formal proof. [*9 marks*]