

## Question 1

[1 + 6 marks]

Consider the following SML function:

```
fun f [] = []
|   f (x::xs) =
    let
        val ys = f xs
    in
        [x]::( (map (fn _ => [x]) ys) @ ys)
    end;
```

- (a) What is the *type* inferred by SML for the above function?
- (b) What is the *result* of function call `f [1,2,3]`? Show all *calls to function f and their result* generated while evaluating `f [1, 2, 3]`.

## Question 2

[3 marks]

Is SML *statically typed* or *dynamically typed*? Briefly explain the differences and the implications for the type system.

### Question 3

[2+2+8 = 12 marks]

Consider a game in which each fighter can be either an archer, a swordsman, a knight, or a hero, and heroes have an associated integer indicating their experience level. Armies can simply be expressed as the list of its fighters. For example,

[Archer, Knight, Archer, Hero(3), Hero(4), Hero(3), Archer] represents an army with 3 archers, 1 knight, 3 heroes, and no swordsmen.

A more compact army representation could be a tuple of the form  $(archers, swordsmen, knights, list\_of\_heroes)$  where the first three elements are integers representing the number of archers, swordsmen and knights, respectively, and *list\_of\_heroes* is a sorted (duplicates not removed) list of the levels of the heroes. For example, the compact version of the previous army would be the tuple:  $(3, 0, 1, [3, 3, 4])$

(a) Define the *datatype* `fighter` whose elements can take the values `Archer`, `Swordsman`, `Knight`, or `Hero(level)` where `level` is an integer.

(b) Define the *type* `army` to represent tuples of the form  $(3, 0, 1, [3, 3, 4])$

(c) Define the SML function

```
compact: fighter list -> army
```

which given call `compact fighter_list` will return the compact representation of the input list of fighters. Furthermore, the function must raise the exception `NoHero` if *the input list contains no heroes*.