Write the *type* of each of the following OCaml expressions in the first blank provided, or *ill-typed* if the expression does not type check. Then, after the ⇒ symbol, write the most simplified *value* of the expression, or leave it blank if it's ill-typed.

```
(a) let a : int list list      = [3::[]]                      ⇒ [[3]]
(b) let b : ill-typed          = [(1::2)::[]]                 ⇒
(c) let c : int list list      = [1::[2;3;4]]                 ⇒ [[1;2;3;4]]
(d) let d : ill-typed          = [[1::2]::[3::4]]             ⇒
(e) let e : int list list      = [[1;2;3;4]::[]]              ⇒ [[1;2;3;4]]
(f) let f : int list list list = [(1::[])::[2]::[3;4]::[]]    ⇒ [[[1];[2];[3;4]]]
(g) let g : ill-typed          = [(5,12,7);(2,5)]             ⇒
(h) let h : int list list      = [[5;12;7];[2;5]]             ⇒ [[5;12;7];[2;5]]
```

Simplify the complex expression below, using step by step substitution. What is the value computed for 'answer' in the following program? (Note: the @ operator glues two lists of the same type together.)

```
    let answer : int list =
      let list = [4;5] in
      let f (num : int) : int list = num :: list in
      let list = [6;7] in
      (f 3) @ list
↦

    let answer : int list =
      let f (num : int) : int list = num :: [4;5] in
      let list = [6;7] in
      (f 3) @ list
↦

    let answer : int list =
      let list = [6;7] in
      (3 :: [4;5]) @ list
↦

    let answer : int list =
      (3 :: [4;5]) @ [6;7]
↦

    let answer : int list =
      [3;4;5] @ [6;7]
↦

    let answer : int list =
      [3;4;5;6;7]
```

Which of the following pieces of code are well-formed OCaml expressions? Refer to the production rules on the back of this sheet. For expressions that do not follow the syntax rules, write *ill-formed*. For those that do, add parentheses and/or underline subexpressions to clarify the boundaries of each expression. The first one had been done for you as an example.

(a) `if (x > 1) then if (y > 2) then 0 else 1 else if (z > 3) then 2 else 3`

`if (x > 1) then (if (y > 2) then 0 else 1) else (if (z > 3) then 2 else 3)`

(b) `let x = 5 in (if (x = r) then 7 else x)`

(c) `if (a = 0) then 9 else if (a = 1) then 8 else if (a = 2) then 7`
Ill-formed (missing else)

(d) `if (a = 0) then 9 else (if (a = 1) then 8 else (if (a = 2) then 7 else 6))`

(e) `if (a = 0) then 9 else if (a = 1) then 8 else 7 else 6`
Ill-formed (too many else)

(f) `if (x < y) then (let z = y-x in z*z) else (let z = x-y in z*z)`

(g) `let b = (if (c = 6) then d else e) in (if (f = b) then g else h)`