Exam set 1

Exercise 1. (Blue.) What is the type of the subexpression `y` as part of the expression below assuming that the whole expression has the type given?

```
(fun double g x -> double (g x)) (fun f y -> f (f y))
 : ('a -> 'b -> 'b) -> 'a -> 'b -> 'b
```

Exercise 2. (Blue.) Write an example function with type:

```
((int -> int) -> bool) -> int
```

Tell "in your words" what it does.

Exercise 3. (Green.) Write a function `last : 'a list -> 'a option` that returns the last element of a list.

Exercise 4. (Green.) Duplicate the elements of a list.

Exercise 5. (Yellow.) Drop every N'th element from a list.

Exercise 6. (Yellow.) Construct completely balanced binary trees of given depth.

In a completely balanced binary tree, the following property holds for every node: The number of nodes in its left subtree and the number of nodes in its right subtree are almost equal, which means their difference is not greater than one.

Write a function `cbal_tree` to construct completely balanced binary trees for a given number of nodes. The function should generate the list of all solutions (e.g. via backtracking). Put the letter `'x'` as information into all nodes of the tree.

Exercise 7. (White.) Due to Yaron Minsky.

Consider a datatype to store internet connection information. The time `when_initiated` marks the start of connecting and is not needed after the connection is established (it is only used to decide whether to give up trying to connect). The ping information is available for established connection but not straight away.

```
type connection_state =
  | Connecting
  | Connected
  | Disconnected

type connection_info = {
  state : connection_state;
  server : Inet_addr.t;
  last_ping_time : Time.t option;
  last_ping_id : int option;
  session_id : string option;
  when_initiated : Time.t option;
  when_disconnected : Time.t option;
}
```

(The types `Time.t` and `Inet_addr.t` come from the library *Core* used where Yaron Minsky works. You can replace them with `float` and `Unix.inet_addr`. Load the Unix library in the interactive toplevel by `#load "unix.cma";;`.) Rewrite the type definitions so that the datatype will contain only reasonable combinations of information.

Exercise 8. (White.) Design an algebraic specification and write a signature for first-in-first-out queues. Provide two implementations: one straightforward using a list, and another one using two lists: one for freshly added elements providing efficient queueing of new elements, and "reversed" one for efficient popping of old elements.

Exercise 9. (Orange.) Implement `while_do` in terms of `repeat_until`.

Exercise 10. (Orange.) Implement a map from keys to values (a dictionary) using only functions (without data structures like lists or trees).

Exercise 11. (Purple.) One way to express constraints on a polymorphic function is to write its type in the form: $\forall\alpha_1...\alpha_n[C].\tau$, where τ is the type of the function, $\alpha_1...\alpha_n$ are the polymorphic type variables, and C are additional constraints that the variables $\alpha_1...\alpha_n$ have to meet. Let's say we allow "local variables" in C: for example $C = \exists\beta.\alpha_1 \doteq \text{list}(\beta)$. Why the general form $\forall\beta[C].\beta$ is enough to express all types of the general form $\forall\alpha_1...\alpha_n[C].\tau$?

Exercise 12. (Purple.) Define a type that corresponds to a set with a googleplex of elements (i.e. $10^{10^{100}}$ elements).

Exercise 13. (Red.) In a height-balanced binary tree, the following property holds for every node: The height of its left subtree and the height of its right subtree are almost equal, which means their difference is not greater than one. Consider a height-balanced binary tree of height h. What is the maximum number of nodes it can contain? Clearly, $\text{maxN} = 2h - 1$. However, finding the minimum number minN is more difficult.

Construct all the height-balanced binary trees with a given nuber of nodes. `hbal_tree_nodes n` returns a list of all height-balanced binary tree with `n` nodes.

Find out how many height-balanced trees exist for `n` = 15.

Exercise 14. (Crimson.) To construct a Huffman code for symbols with probability/frequency, we can start by building a binary tree as follows. The algorithm uses a priority queue where the node with lowest probability is given highest priority:

1. Create a leaf node for each symbol and add it to the priority queue.
2. While there is more than one node in the queue:
    a. Remove the two nodes of highest priority (lowest probability) from the queue.
    b. Create a new internal node with these two nodes as children and with probability equal to the sum of the two nodes' probabilities.
    c. Add the new node to the queue.
3. The remaining node is the root node and the tree is complete.

Label each left edge by `0` and right edge by `1`. The final binary code assigns the string of bits on the path from root to the symbol as its code.

We suppose a set of symbols with their frequencies, given as a list of Fr(S,F) terms. Example: `fs = [Fr(a,45); Fr(b,13); Fr(c,12); Fr(d,16); Fr(e,9); Fr(f,5)]`. Our objective is to construct a list `Hc(S,C)` terms, where `C` is the Huffman code word for the symbol `S`. In our example, the result could be `hs = [Hc(a,'0'); Hc(b,'101'); Hc(c,'100'); Hc(d,'111'); Hc(e,'1101'); Hc(f,'1100')]` [`Hc(a,'01')`,...etc.]. The task shall be performed by the function huffman defined as follows: `huffman(fs)` returns the Huffman code table for the frequency table `fs`.

Exercise 15. (Black.) Implement the Gaussian Elimination algorithm for solving linear equations and inverting square invertible matrices.