1995 Paper 1 Question 3

Foundations of Computer Science

This question has been translated from Standard ML to OCaml

An OCaml list can be considered to be a set if it has no repeated elements, e.g., `[4; 2; 3]` is a set but `[4; 2; 4; 3]` is not.

Write an OCaml function `intersect` to compute the set-theoretic intersection of two lists that satisfy this property of being a set. (The intersection of two sets is the set of elements that appear in both sets.) Your function must also satisfy conditions (a)–(c) below:

(a) The result list has no repeated elements;

(b) The number of cons (`::`) operations performed does not exceed the number of elements in the result list;

(c) The elements of the result list appear in the same order as they do in the first argument list.

You may assume the existence of `[]` (empty list) and `::` (cons operation). All other functions over lists must be defined by you. Little credit will be given for answers that do not satisfy conditions (a)–(c).

Write an OCaml function `subtract` that given two lists satisfying the property of being a set, returns a list consisting of those elements of the first list that do not appear in the second list. Your subtract function must satisfy conditions (a)–(c) above.

Write an OCaml function `union` that given two lists satisfying the property of being a set, returns a list consisting of those elements that appear in one or other or both of the lists. Your union function must satisfy conditions (a) and (b) above and *(d)* below:

(d) Elements from the first argument list appear before any others in the result and in the same order as they appear in the first argument.

State the most general type of your union function.

[10 marks]

1995 Paper 1 Question 5

Foundations of Computer Science

This question has been translated from Standard ML to OCaml

Describe and compare the call-by-value, call-by-name, and call-by-need evaluation strategies for functional programming languages.

The OCaml function `butlast` removes the last element from a non-empty list:

```
exception Butlast
let rec butlast = function
  | [] -> raise Butlast
  | [_] -> []
  | (x::xs) -> x::(butlast xs)
```

Show how the evaluation of `butlast [[1; 2]; []; [3]; [4; 5]]` proceeds in OCaml.

Write an iterative version of `butlast` (i.e. one in which the recursive function calls are tail recursive). You may assume the existence of the append (`@`) function.

State with justification the time complexity of your function.

An OCaml variant type of lazy lists can be defined by:

```
type 'a lazy_list = Nil | Cons of unit -> 'a * 'a lazy_list
```

An 'infinite' list of increasing integers can be generated by the function `infinite` below:

```
let rec infinite n = Cons (fun () -> (n, infinite (n + 1)))
```

Write a version of `butlast` for lazy lists which terminates when applied to an infinite lazy list such as `infinite 0`.

Can an iterative version of this function be written that still terminates on infinite lazy lists? Explain your reasoning.

[20 marks]

1995 Paper 1 Question 6

Foundations of Computer Science

This question has been translated from Standard ML to OCaml

Consider the following OCaml declarations:

```
type bit = One | Zero
type cardinal = Cardinal of bit list
exception Result_would_be_negative
```

Define functions to add, subtract and multiply `cardinal`s, viewing them as representations of integers stored as in binary with the least significant bit of the value first in the list. Thus for instance the number eleven has the value 1011 in binary and so would be represented (note the ordering of the bits) by the structure

```
Cardinal [One; One; Zero; One]
```

Using Big-O notation state the time complexity of your add and multiply functions.

[20 marks]

1996 Paper 1 Question 2

Foundations of Computer Science

This question has been translated from Standard ML to OCaml

The OCaml variant type `bool`, defined below, is to be used to represent boolean expressions.

```
type bool = Var of string
          | Not of bool
          | And of bool * bool
          | Or  of bool * bool
```

The constructor `Var` is used to represent named boolean variables, and `Not`, `And` and `Or` are used to represent the corresponding boolean operators.

Define a function that will return a list of the distinct names used in a given boolean expression. [4 marks]

A context is represented by a list of strings corresponding to the boolean variables that are set to true. All other variables are deemed to be set to false. Define a function that will evaluate a given boolean expression in a given context. [3 marks]

Incorporate your two functions into a program that will determine whether a given boolean expression is true for all possible settings of its variables. [3 marks]

1996 Paper 1 Question 5

Foundations of Computer Science

This question has been translated from Standard ML to OCaml

Define an OCaml function `rotations` that will compute the list of all rotations of a given list. For example

```
rotations [1; 2; 3] = [[1; 2; 3]; [2; 3; 1]; [3; 1; 2]]
```

The order in which the rotations occur is unimportant. [10 marks]

Carefully explain how your function works and estimate the time complexity of your solution. [10 marks]

1996 Paper 1 Question 6

Foundations of Computer Science

This question has been translated from Standard ML to OCaml

Give the declaration of an OCaml variant type that could be used in the representation of a lazy list of integers, and illustrate its use by defining a function `ints` that when given an argument `n` yields a lazy list of integers from `n` to infinity. [5 marks]

The decimal representation of a real number in the range 0 to 1 is to be represented as an infinite sequence of the decimal digits following the decimal point ($0.d_1d_2...$). Define a function `mknumb` which when applied to the digit function `dig` will construct a lazy list of these digits where the ith digit (d_i) is given by `dig i`. [5 marks]

Suppose we have an infinite sequence of such numbers $[r_1, r_2, ...]$, in which the digits of the decimal expansion of r_i are given by the digit function f_i, and that the collection of digit functions is represented by the lazy list $[f_1, f_2, ...]$. Define suitable types for the list of numbers and the list of digit functions. [5 marks]

Define a function `newnumb` which when given the lazy list of digit functions will yield a lazy list of digits that have the property that its ith digit differs from the ith digit of r_i. [5 marks]

1997 Paper 1 Question 1

Foundations of Computer Science

This question has been translated from Standard ML to OCaml

The variant type `pri` defined below is to be used for the representation of priority queues which are finite or infinite ordered sets of integers.

```
type pri = E
         | N of int * (unit -> pri)
```

Define an OCaml function `intfromto i j : int -> int -> pri` which will return a representation of the ordered set of integers

```
{ i, i + 1, ..., j }
```

Define the function `first p : pri -> int` that will return the first (and hence smallest) integer in the given queue `p`, and `rest p : pri -> pri` that will return (if possible) a representation of the given queue `p` with its smallest element removed. Your implementation should be such that the expression

```
first (rest (intsfromto 20 1000000))
```

should evaluate efficiently.

Define an OCaml function `ins i p : int -> pri -> pri` which will return a priority queue with the integer `i` inserted in the proper position of the given queue `p`.

[10 marks]

1997 Paper 1 Question 5

Foundations of Computer Science

This question has been translated from Standard ML to OCaml

Noughts and Crosses is a game played by two players (O and X) on a board with nine positions numbered as follows:

1	2	3
4	5	6
7	8	9

The players place their marks (O and X) in unoccupied positions on the board until the game is complete. A completed game is when either

(a) there is a straight line of three Xs giving a win for X, *or*

(b) there is a straight line of three Os giving a win for O, *or*

(c) all nine positions are occupied, in which case the game is drawn.

O is the first player to move.

It is required to construct an OCaml structure representing the tree of all possible games. Each node of the tree should represent a reachable board state, with the root being the empty board, and the leaf nodes corresponding to won, lost or drawn games.

Define the OCaml variant type `tree` that you would use to represent this game tree. [3 marks]

Define the function `mktree : unit -> tree` to construct the complete game tree, explaining carefully how it works. There is no need for your implementation to be efficient in either space or time. [10 marks]

Briefly discuss ways in which your implementation of `mktree` could be made more efficient. [4 marks]

Define a function `winner_is_O : tree -> int` which when applied to the complete tree will yield the number of distinct games in which O wins. [3 marks]

1997 Paper 1 Question 6

Foundations of Computer Science

This question has been translated from Standard ML to OCaml

A rooted directed graph has vertices identified by integers. Each vertex `v` has a left successor given by `left v` and a right successor given by `right v`, where `left` and `right` are OCaml functions of type `int -> int`. The graph contains the root and all vertices reachable by paths from the root.

Define a variant type `graph` that could be used to represent such a graph with given root, and left and right functions, and define a function `mkgraph root left right` that can create values of type `graph`. Show that such values can be used to represent both finite and infinite graphs. [4 marks]

A path through the graph is represented by a `bool list` with `true` and `false` indicating left and right edges, respectively.

Define the function `last : graph -> bool list -> int` that will yield, for a given graph, the identity of the vertex reached by following the given path from the root. [3 marks]

In a new application, where `last` is repeatedly called, it is required for it to return both the identity of the last vertex and a count of how often this particular vertex has been returned. Define a new version of the variant type `graph`, containing mutable values, that could be used. [3 marks]

Illustrate the use of this type by defining the new versions of `mkgraph` and `last`. [10 marks]

1998 Paper 1 Question 1

Foundations of Computer Science

This question has been translated from Standard ML to OCaml

Code the function `least n xs`, which returns the least n elements of the list xs of real numbers. The result does not have to be sorted. You may assume $n \leqslant$ `List.length` xs.

[Hint: Function `least` is a simple modification of quicksort. A solution that works by sorting the entire list will receive little credit. You may take the function `List.length` as given.]

[10 marks]

1998 Paper 1 Question 5

Foundations of Computer Science

This question has been translated from Standard ML to OCaml

Discuss how lazy lists and mutable lists can be coded in OCaml. How do they compare with OCaml's built-in lists? Illustrate your answer by considering the operations of reversing a list and of concatenating two lists. Your discussion should mention the main programming hazards. [6 marks]

The function `odds` is to return the list of alternate elements of its input. For example, `odds [`a`; `b`; `c`; `d`; `e`] = [`a`; `c`; `e`]` and `odds [`a`; `b`] = [`a`]`. Code `odds` using

(a) ordinary OCaml lists [3 marks]

(b) lazy lists [5 marks]

(c) mutable lists (as an imperative operation — so that `odds` has type `'a mlist -> unit` for a suitable variant type `'a mlist` of mutable lists, to be defined) [6 marks]

1998 Paper 1 Question 6

Foundations of Computer Science

This question has been translated from Standard ML to OCaml

What does $O(g(n))$ mean, and what is its relevance to programming? (Describe both advantages and limitations.) [5 marks]

Consider the following OCaml declarations, for tree-like expressions:

```
type 'a expr = Join of 'a expr * 'a expr
             | Tip  of 'a

let rec flatten = function
  | Tip x         -> [x]
  | Join (e1, e2) -> flatten e1 @ flatten e2
```

The *size* of an expression is the number of `Tip`s it contains. State the complexity of `flatten` e, measured in cons operations, as a function of the size of e:

(a) in the worst case [3 marks]

(b) in the average case [4 marks]

(c) in the best case [3 marks]

Code a function `flat` such that `flat` $e =$ `flatten` e for all e, justifying this claim. Show that `flat`'s worst-case complexity is linear. [5 marks]

1999 Paper 1 Question 1

Foundations of Computer Science

This question has been translated from Standard ML to OCaml

A *permutation* of a list is any list that can be derived from it by re-arranging the elements. Write an OCaml function that returns the list of all the permutations of its argument. Explain your code clearly and carefully.

For example, applied to the list `[1; 2; 3]`, your function should return the list whose elements are `[1; 2; 3]`, `[2; 1; 3]`, `[2; 3; 1]`, `[1; 3; 2]`, `[3,1,2]` and `[3,2,1]`.

You may assume that the elements of the argument are distinct. The elements of the result may appear in any order.

[10 marks]

1999 Paper 1 Question 5

Foundations of Computer Science

This question has been translated from Standard ML to OCaml

"We face the Year 2000 crisis because programmers did not apply the principles of data abstraction." Discuss. [4 marks]

Your employer asks you to implement a dictionary. The pattern of usage will consist of taking an empty dictionary and performing many insertions and lookups. You must choose one of three data structures. Each requires $O(\log n)$ time for the lookup and update operations, where n is the number of items in the dictionary. They are (1) binary search trees, which take $O(\log n)$ time in the average case; (2) balanced trees, which need complicated algorithms but take $O(\log n)$ time in the worst case; (3) self-adjusting trees, which take $O(\log n)$ amortised time in the worst case.

Explain the differences between the three notions of $O(\log n)$ time. Argue that any of the three data structures might turn out to be the best, depending upon further details of the application. If no further details are available, which of the three is the safest choice? [8 marks]

An algorithm requires $T(n)$ units of space given an input of size n, where the function T satisfies the recurrence

$$\begin{aligned} T(1) &= 1 \\ T(n) &= T(n/2) + n \qquad (n > 1). \end{aligned}$$

Express the algorithm's space requirement using O-notation, carefully justifying your answer. [8 marks]

1999 Paper 1 Question 6

Foundations of Computer Science

This question has been translated from Standard ML to OCaml

Describe OCaml's facilities for treating functions as data, giving examples of their use in programs. Illustrate your answer by discussing the function `fold_right`:

```
let rec fold_right f l e
  match l with
  | [] -> e
  | x::xs -> f x (fold_right f xs e)
```

[7 marks]

You have been asked to implement a data structure to represent family relationships. For each person, it should record his or her name, mother, father, and children. As a first attempt, you have been given the following variant `type` declaration:

```
type person = Person of string * person * person * person list
```

Identify two problems with this declaration that make it unusable. Modify the declaration to correct these problems. [6 marks]

Consider the following, simpler data structure for associating a person with his or her children:

```
type famtree = B of string * famtree list
```

Write an OCaml function that takes two arguments: a predicate P over family trees (a function of type `famtree -> bool`) and a family tree t. The result should be the list of all subtrees of t (possibly including t itself) satisfying the predicate. For full credit, give due attention to efficiency. [7 marks]

2000 Paper 1 Question 1

Foundations of Computer Science

This question has been translated from Standard ML to OCaml

Code the curried function `exf`, which takes as arguments the function `f` and the list `l`. The result must consist of those elements x of `l` such that `f` x is also a member of `l`. The elements of the result must be distinct from each other but may appear in any order. For example, if `f x = x + 1` and `l = [9; 3; 2; 2; 8]` then the result should be `[2; 8]` or `[8; 2]`. [9 marks]

State, with justification, the type of `exf`. [1 mark]

2000 Paper 1 Question 5

Foundations of Computer Science

This question has been translated from Standard ML to OCaml

State the time complexity of the *lookup* and *update* operations for each of the following:

(*a*) association lists

(*b*) binary search trees

(*c*) functional arrays (implemented as binary trees)

Use O-notation and include both the average-case and worst-case complexity. [6 marks]

You are provided with the OCaml code for binary search trees, including the *lookup* and *update* operations. Use these operations to code a sorting function that works by repeatedly inserting elements of a list into a binary search tree, then converting the final binary search tree back into a list. [4 marks]

Consider the following methods of sorting a list:

(*a*) Locate the smallest element of the input. The output is this element followed by the result of recursively sorting the remaining elements.

(*b*) Take the first 16 elements of the input and sort them using special hardware. Sort the remaining elements recursively. The output is the result of merging the two sorted lists.

(*c*) Take the first 20% of the input elements. Sort them and the remaining 80% recursively. The output is the result of merging the two sorted lists.

For each of these methods, state with justification the worst-case complexity in terms of the number of comparisons. [10 marks]

2000 Paper 1 Question 6

Foundations of Computer Science

This question has been translated from Standard ML to OCaml

Describe how OCaml lists are represented in storage. Your answer should include diagrams illustrating how the representation of `[a; b] @ [c; d]` is derived from those of the lists `[a; b]` and `[c; d]`, indicating any sharing of memory. How efficient is the evaluation of `[a; b] @ l` if the list `l` is very long? [4 marks]

What are *cyclic lists* and how can they be created in OCaml? [2 marks]

NB This question was unambiguous in Standard ML, which doesn't have OCaml's recursive value definitions.

Describe OCaml's reference types and their applications. In particular, compare mutable data structures with ordinary OCaml types. [6 marks]

Code an OCaml function that takes a mutable list and returns true if the list is cyclic, otherwise returning false. Explain why your function is correct.
[Hint: in OCaml, the physical equality test p `==` q is true if p and q refer to the same location in memory.] [8 marks]

2001 Paper 1 Question 1

Foundations of Computer Science

This question has been translated from Standard ML to OCaml

(*a*) An OCaml program makes the following declarations:

```
let x = ref 0

let f n = (x := !x + 1; n + !x)

let g n =
  let x = ref 0 in
  x := !x + 1; n + !x
```

Consider evaluating each of the following expressions:

```
let l1 = List.map f [1; 2; 3; 4]
let l2 = List.map g [1; 2; 3; 4]
let l3 = List.map ref [5; 5; 5]
```

What value is returned in each case and how are the references affected? [5 marks]

(*b*) Code the function `filter` such that `filter p xs` returns the list of those elements of the list `xs` satisfying the predicate `p`. [1 mark]

(*c*) Use `filter` to express Quicksort. [4 marks]

2001 Paper 1 Question 5

Foundations of Computer Science

This question has been translated from Standard ML to OCaml

(*a*) Write brief notes on the following:

(*i*) Re-coding a function to make it iterative. [4 marks]

(*ii*) The difference between depth-first and breadth-first search. [4 marks]

(*iii*) OCaml's exception-handling facilities. [4 marks]

(*b*) Code a function whose input is a list of integers and whose output is the list of all the integers that can be expressed as the sum of one or more of the supplied integers. For example, given `[1; 5; 10]` a correct output is `[1; 5; 10; 6; 11; 15; 16]`; the order of the elements in the output does not matter. [5 marks]

(*c*) Modify the function that you coded above so that the elements of its output appear in numerical order and without repetitions. [3 marks]

2001 Paper 1 Question 6

Foundations of Computer Science

This question has been translated from Standard ML to OCaml

To represent the power series $\sum_{i=0}^{\infty} a_i\, x^i$ in a computer amounts to representing the coefficients a_0, a_1, a_2, ... One possible representation is by a function of type `int->real` that returns the coefficient a_i given i as an argument. An alternative representation is the following type:

```
type power = Cons of float * (unit -> power)
```

(a) Demonstrate the two representations by using each of them to implement these two power series:

(i) The constant power series c, with $a_0 = c$ and $a_i = 0$ for $i > 0$. [3 marks]

(ii) The Taylor series $\sum_{i=0}^{\infty} x^i/i!$ for the exponential function. [4 marks]

(b) Also implement (using both representations) each of the following operations on power series:

(i) Product with a scalar, given by $c \cdot \left(\sum_{i=0}^{\infty} a_i\, x^i\right) = \sum_{i=0}^{\infty} (c a_i)\, x^i$. [3 marks]

(ii) Sum, given by $\left(\sum_{i=0}^{\infty} a_i\, x^i\right) + \left(\sum_{i=0}^{\infty} b_i\, x^i\right) = \sum_{i=0}^{\infty} (a_i + b_i)\, x^i$. [4 marks]

(iii) The product $\left(\sum_{i=0}^{\infty} a_i\, x^i\right) \times \left(\sum_{i=0}^{\infty} b_i\, x^i\right)$, where the ith coefficient of the result is $a_0\, b_i + a_1\, b_{i-1} + \cdots + a_i\, b_0$. [6 marks]

You may assume the OCaml function `float_of_int` of type `int -> float` that maps an integer to the equivalent real number.

2002 Paper 1 Question 1

Foundations of Computer Science

This question has been translated from Standard ML to OCaml

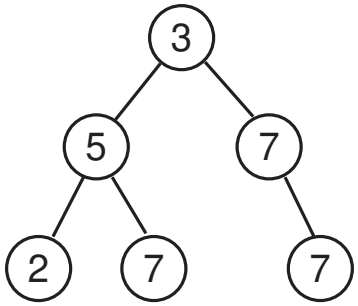
Consider the following OCaml declarations, involving binary trees:

```
type 'a tree = Lf
             | Br of 'a * 'a tree * 'a tree

exception E

let rec path = function
  | Lf -> raise E
  | Br(v, t1, t2) ->
      try
        if v = 7 then []
        else 1 :: path t1
      with E -> 2 :: path t2
```

(*a*) The function `path` returns a path (a list of 1s and 2s) to an occurrence of the number 7 in the tree. Carefully explain how `path` works, taking the tree shown below as an example and indicating which occurrence of 7 will be found. [5 marks]



(*b*) Code the function `paths`, which returns the list of all paths to 7s in a binary tree. [5 marks]

2002 Paper 1 Question 5

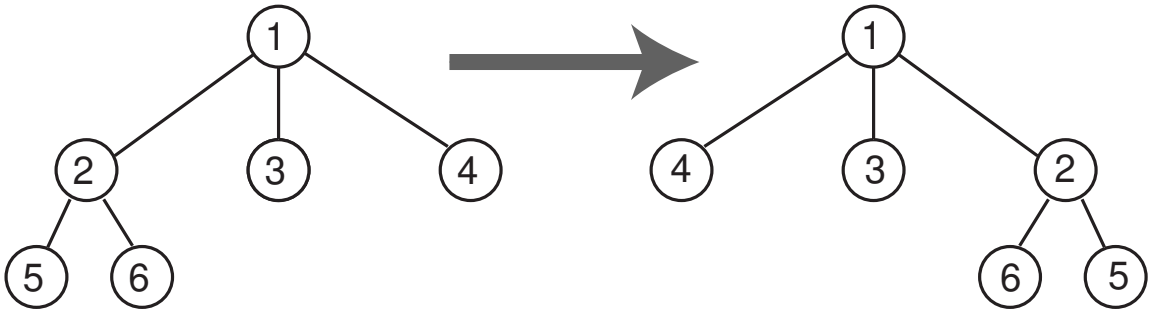
Foundations of Computer Science

This question has been translated from Standard ML to OCaml

This question concerns the following OCaml declaration of a tree type:

```
type 'a fan = Wave of 'a * ('a fan) list
```

(*a*) Declare the function `flip`, which maps a tree to a mirror image of itself, as illustrated: [3 marks]



(*b*) Declare the curried function `paint f`, which copies a tree while applying the function `f` to each of its labels. [3 marks]

(*c*) Declare the function `same_shape`, which compares two trees and returns `true` if they are equal except for the values of their labels and otherwise returns `false`. [5 marks]

(*d*) State the types of functions `flip`, `paint` and `same_shape`. [3 marks]

(*e*) The function `paper` is declared in terms of the familiar functional `fold_right`:

```
let rec fold_right f l e =
  match l with
  | [] ->  e
  | x::xs -> f x (fold_right f xs e)

let rec paper (Wave(x, fs)) q = fold_right paper fs (q + 1)
```

Describe the computation that results when `paper` is applied to a tree. [6 marks]

2002 Paper 1 Question 6

Foundations of Computer Science

This question has been translated from Standard ML to OCaml

(*a*) Explain how O-notation is used to express efficiency of algorithms. [5 marks]

(*b*) Arrange the following list of complexity classes in order of decreasing efficiency in n. Briefly justify each relationship.

$$O(5n^2) \qquad O(e^n) \qquad O(n^{1/3}) \qquad O(n^3 - 3n^2) \qquad O(\log n) \qquad O(n2^n)$$

[4 marks]

(*c*) Suppose that f is a function from integers to integers such that $i \leqslant j$ implies $f(i) \leqslant f(j)$. Then there is an efficient algorithm to solve the equation $f(k) = y$, given the desired y and a range of values in which to search for k: the idea is repeatedly to halve this range. Code this algorithm as the OCaml function `search` whose arguments are f, y, and the range (a, b). Its result should be the greatest k such that $f(k) \leqslant y$ and $a \leqslant k \leqslant b$, provided such a k exists.
[11 marks]

2003 Paper 1 Question 1

Foundations of Computer Science

This question has been translated from Standard ML to OCaml

(*a*) Carefully explain the concept of a curried function, with examples. [4 marks]

(*b*) Explain the purpose and operation of the function `f`:

```
let rec fold_right f l e =
  match l with
  | [] ->  e
  | x::xs -> f x (fold_right f xs e)

fun f xs =
  fold_right (fun (x, y) (xs, ys) -> (x::xs, y::ys)) xs ([], [])
```

[4 marks]

(*c*) Write a function that accepts a list and returns the list consisting of its first, second, fourth, fifth, seventh, eighth, tenth, etc., elements. In other words, your function should return all the elements except those whose position in the input list is a multiple of three. [2 marks]

2003 Paper 1 Question 5

Foundations of Computer Science

This question has been translated from Standard ML to OCaml

(*a*) Describe how lazy lists can be implemented using OCaml. [2 marks]

(*b*) Code a function to concatenate two lazy lists, by analogy to the 'append' function for ordinary OCaml lists. Describe what happens if your function is applied to a pair of infinite lists. [3 marks]

(*c*) Code a function to combine two lazy lists, interleaving the elements of each. [3 marks]

(*d*) Code the lazy list whose elements are all ordinary lists of zeroes and ones, namely `[]; [0]; [1]; [0; 0]; [0; 1]; [1; 0]; [1; 1]; [0; 0; 0]; ...`. [6 marks]

(*e*) A *palindrome* is a list that equals its own reverse. Code the lazy list whose elements are all palindromes of 0s and 1s, namely `[]; [0]; [1]; [0; 0]; [0; 0; 0]; [0; 1; 0]; [1; 1]; [1; 0; 1]; [1; 1; 1]; ...`. You may take the reversal function `List.rev` as given. [6 marks]

2003 Paper 1 Question 6

Foundations of Computer Science

This question has been translated from Standard ML to OCaml

(*a*) Explain in detail how a binary search tree represents a dictionary. If reference types are not used, what is the average-case cost of an update operation? [8 marks]

(*b*) A *mutable* binary tree is either a leaf or a branch node containing a label and references to two other mutable binary trees. Present the OCaml variant type declaration for mutable binary trees. [2 marks]

(*c*) Describe how to use this type to implement (mutable) binary search trees. Give OCaml code for the lookup and update operations. The update operation must never copy existing tree nodes. [10 marks]

COMPUTER SCIENCE TRIPOS Part IA – 2004 – Paper 1

1 Foundations of Computer Science (ACN)

This question has been translated from Standard ML to OCaml

(*a*) What does the OCaml function `map` do? Give an example, first coded without `map` and then with it, to illustrate how it can lead to more compact or comprehensible code. [3 marks]

(*b*) Functions `fold_left` and `fold_right` might be defined as

```
let rec fold_left f e = function
  | [] -> e
  | x::xs -> fold_left f (f ex) xs

let rec fold_right f l e = match l with
  | [] -> e
  | x::xs -> f x (fold_right f xs e)
```

Explain what these two functions do and why they may be useful. [4 marks]

(*c*) Here is a typical use of `map`:

```
let mangle n = (n - 2) * (n + 7)
let mangle_list x = map mangle x
```

Show how to express `mangle_list` using one of the "fold" functions rather than `map`. [3 marks]

COMPUTER SCIENCE TRIPOS Part IA – 2004 – Paper 1

5 Foundations of Computer Science (ACN)

This question has been translated from Standard ML to OCaml

The following OCaml type can be viewed as defining a lazy or infinite sort of tree where each node in the tree holds an integer:

```
type tr = N of int * unit -> tr * unit -> tr
```

(*a*) Write a function called `ndeep` such that if `n` is an integer and `z` is a tree (i.e. of type `tr`) the call `ndeep n z` will return an ordinary list of all the 2^n integers at depth exactly `n` in the tree. Note that if `n` = 0 it will return a list of length 1, being just the top integer in the tree. Comment on its efficiency. [8 marks]

(*b*) You are given a `tr`, and told that it contains arbitrarily large values at least somewhere in it. You want to find a value from it that is bigger than 100 (but if there are many big values it does not matter which one is returned). Because the tree is infinite you cannot use simple depth-first search: you decide to use iterative deepening. Thus you first check all integers at depth 1, then at depth 2, depth 3, ... and return when you first find a value that is greater than 100.

Use exception handling to return the large value when you find it. Present and explain code that searches the lazy tree. [12 marks]

COMPUTER SCIENCE TRIPOS Part IA – 2004 – Paper 1

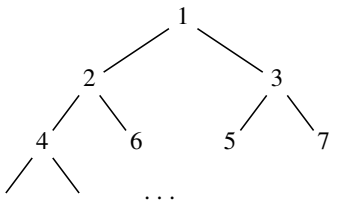
6 Foundations of Computer Science (ACN)

This question has been translated from Standard ML to OCaml

In OCaml it is possible to use functions as values: they can be passed as arguments and returned as results. Explain the notation used to write a function without having to give it a name. [2 marks]

This question looks at two different ways of implementing functional arrays.

(*a*) One possible functional implementation of an array is based on trees, and the path to a stored value follows the binary code for the subscript:



where in the above diagram the numbers show where in the tree a value with the given subscript will live.

Write code that creates, retrieves values from and updates an array that has this representation, and using big-O notation explain the associated costs. [8 marks]

(*b*) A different way of handling functional arrays is to represent the whole array directly by a function that maps from integers to values. To access the item at position k in such an array you just use the array as a function and give it k as its argument, and to update the array you need to create a new function reflecting the changed value.

If the array is to hold integer values, what OCaml type does it have? [1 mark]

Write a function `update a n v` where `a` is a functional array in this style, `n` is an integer index and `v` is a new value. The result of the call to `update` must behave as an array that stores all the values that `a` did except that it is as if an assignment of the style "`a[n] := v`" has been performed. [5 marks]

In big-O notation, what is the cost of your `update` function? After a sequence of updates what is the cost of accessing the array? [4 marks]

COMPUTER SCIENCE TRIPOS Part IA – 2005 – Paper 1

1 Foundations of Computer Science (LCP)

This question has been translated from Standard ML to OCaml

It is required to implement polynomials, which may contain any number of different variables, with the operations of *addition* and *equality test*.

- Computing that the sum of $xy^2 + 2xz - y$ and $y + 4xw - xz$ is $xz + xy^2 + 4xw$ is an example of polynomial addition.

- Determining that $xy + z^2$ equals $zz + yx$ is an example of an equality test.

(a) Design a suitable representation of polynomials, in OCaml. Justify the choices you make. [6 marks]

(b) Describe how you would implement addition and the equality test, and express their efficiency using O-notation. It is not necessary to present OCaml code. [4 marks]

COMPUTER SCIENCE TRIPOS Part IA – 2005 – Paper 1

5 Foundations of Computer Science (LCP)

This question has been translated from Standard ML to OCaml

(*a*) Explain the operation of the Quicksort algorithm. Illustrate your answer by applying it to the list `[8; 3; 6; 12; 2; 9; 20; 1; 5; 0; 7; 13; 4; 11; 10]`. [6 marks]

(*b*) Write an OCaml function for finding the median of three integers. [3 marks]

(*c*) A variant of Quicksort uses a novel method of choosing the pivot element. Instead of using the head of the list, it uses the median of the first, middle and last elements of the list. Express this algorithm in OCaml. You may assume the existence of the function `length`, but you may not assume that the items being sorted are distinct. [7 marks]

(*d*) What is the average-case execution cost, measured in terms of the number of comparisons, for the version of Quicksort described in part (*c*) above? Justify your answer carefully. [4 marks]

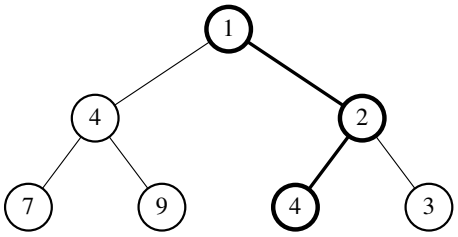
6 Foundations of Computer Science (LCP)

This question has been translated from Standard ML to OCaml

Consider a variant type of binary trees where both leaves and branches carry labels:

```
type 'a tree = Twig of 'a
             | Br of 'a * 'a tree * 'a tree
```

A *path* in a binary tree is a series of labels proceeding from the root to a leaf, as shown in the diagram:



Consider the problem of finding a path in a binary tree such that the integer sum of the labels satisfies a given property. (In the example above, the highlighted path sums to a prime number.)

(*a*) Write an OCaml function `find_path` such that `find_path p t` returns some path in `t` whose sum satisfies the boolean-valued function `p`. If no such path exists, the function should raise an exception. [5 marks]

(*b*) Write an OCaml function `all_paths` such that `all_paths p t` returns the list of all paths in `t` whose sums satisfy the boolean-valued function `p`. [6 marks]

(*c*) Write an OCaml function `all_pathq` that is analogous to `all_paths` but returns a lazy list of paths. For full credit, your function should find paths upon demand rather than all at once. [Hint: try adding solutions to an accumulating argument.] [9 marks]

COMPUTER SCIENCE TRIPOS Part IA – 2006 – Paper 1

1 Foundations of Computer Science (LCP)

This question has been translated from Standard ML to OCaml

Give an example of an OCaml function belonging to *each* of the following complexity classes:

(*a*) $O(1)$;

(*b*) $O(n)$;

(*c*) $O(n \log n)$;

(*d*) $O(n^2)$;

(*e*) $O(2^n)$.

Each answer may contain code fragments (involving well-known functions) rather than self-contained programs, but must include justification. (The upper bound in each case should be reasonably tight.) [2 marks each]

COMPUTER SCIENCE TRIPOS Part IA – 2006 – Paper 1

5 Foundations of Computer Science (LCP)

This question has been translated from Standard ML to OCaml

(*a*) This question concerns the data structure of queues.

(*i*) Describe the primitive queue operations. [3 marks]

(*ii*) Describe an efficient implementation of queues, presenting code fragments as appropriate (a complete program listing is not required). [3 marks]

(*iii*) Carefully discuss the efficiency of your implementation, using the concept of amortised time. [4 marks]

(*b*) Write an OCaml function to compute all permutations of its argument, a list. (You may assume that the elements of this list are distinct.) For example, given the argument `[1; 2; 3]`, the result should be a list consisting of the elements `[1; 2; 3]`, `[2; 1; 3]`, `[2; 3; 1]`, `[1; 3; 2]`, `[3; 1; 2]` and `[3; 2; 1]` in any order. For full credit, your code must be well structured and clearly explained.
[10 marks]

COMPUTER SCIENCE TRIPOS Part IA – 2006 – Paper 1

6 Foundations of Computer Science (LCP)

This question has been translated from Standard ML to OCaml

(*a*) Contrast *ordinary lists*, *lazy lists* and *mutable lists* by

(*i*) presenting the OCaml type declaration of each type of list, and [3 marks]

(*ii*) implementing a filter functional for each type of list. [7 marks]

(The mutable version should remove the elements that do not satisfy the given predicate, rather than constructing a new list.)

(*b*) The *intersection* of two dictionaries is the largest dictionary that agrees with them both. For example, if one dictionary is cat=3, dog=2, rabbit=9 while the other is cat=4, dog=2, hamster=9, then their intersection is dog=2.

Code an OCaml function to compute the intersection of two dictionaries, where dictionaries are represented by binary search trees. You may assume that the dictionary lookup and update operations are provided. For full credit, your solution must be simple and clear. [10 marks]

COMPUTER SCIENCE TRIPOS Part IA – 2007 – Paper 1

1 Foundations of Computer Science (LCP)

This question has been translated from Standard ML to OCaml

(*a*) Code the OCaml function `merge`, which combines two ordered lists to form an ordered list containing the elements of both. [2 marks]

(*b*) Code a top-down merge sort function in OCaml. You may assume that basic functions on lists are given, provided you describe them briefly. [3 marks]

(*c*) State the time complexities of merge and merge sort, justifying your answer carefully. [5 marks]

COMPUTER SCIENCE TRIPOS Part IA – 2007 – Paper 1

5 Foundations of Computer Science (LCP)

This question has been translated from Standard ML to OCaml

(*a*) Consider the following piece of OCaml code:

```
type 'a tree = Lf | Br of 'a * 'a tree * 'a tree
exception Blair

let rec tony p = function
  | Lf -> true
  | Br (x,t1,t2) -> if not (p x) then raise Blair
                    else try tony p t1 with Blair -> tony p t2

let gordon p t = try tony p t with Blair -> false
```

(*i*) Code a function that returns the same results as `gordon` but makes no use of exceptions. [4 marks]

(*ii*) What property of binary trees does `gordon` express? [3 marks]

(*b*) Write brief notes on the OCaml type `exn`. [3 marks]

(*c*) Consider the following piece of OCaml code:

```
type 'a result = Ian of 'a | Cherie of exn

let what f x = try Ian (f x) with e -> Cherie e
```

We ask OCaml to evaluate the expression

```
map (what (tony (fun x -> x <> 0))) [ta; tb]
```

and the response is as follows:

```
- : bool result list = [Ian true; Cherie Blair]
```

What is the type of `what (tony (fn x => x <> 0))`, and what can we infer about the binary trees `ta` and `tb`? Justify both answers carefully. [5+5 marks]

6 Foundations of Computer Science (LCP)

This question has been translated from Standard ML to OCaml

(*a*) Write brief notes on reference types in OCaml and on control structures for imperative programming. [6 marks]

Consider the following OCaml type:

```
type 'a meal = Snack of 'a
             | Lunch of 'a meal * 'a meal
             | Feast of 'a meal * 'a meal * 'a meal
```

(*a*) Write a function that is equivalent to `snacker` below but makes no use of references. Briefly explain why the two functions are equivalent.

```
let snacker m =
  let l = ref [] in
  let munch = function
    | Snack x -> (l := x :: !l)
    | Lunch (m1, m2) -> (munch m1; munch m2)
    | Feast (m1, m2, m3) -> (munch m1; munch m2; munch m3)
  in
  munch m; !l
```

[5 marks]

(*b*) Write a function `gluttony` such that `gluttony m1 m2` makes a copy of `m1`, replacing every `Snack` node with `m2`. [3 marks]

(*c*) Write a function `glut` such that `glut k m1 m2` makes a copy of `m1`, replacing the kth `Snack` node with `m2`. Nodes are counted from left to right, with the leftmost node being number one. [6 marks]

COMPUTER SCIENCE TRIPOS Part IA – 2008 – Paper 1

1 Foundations of Computer Science (LCP)

This question has been translated from Standard ML to OCaml

(*a*) *This part concerned a feature of Standard ML which does apply to OCaml.* [3 marks]

(*b*) The list l_2 is a *sublist* of l provided there exist lists l_1 and l_3 such that $l = l_1@l_2@l_3$. Write an OCaml function `sublist` to test whether one list is a sublist of another one. Include a clear explanation of how your code works. [7 marks]

COMPUTER SCIENCE TRIPOS Part IA – 2008 – Paper 1

5 Foundations of Computer Science (LCP)

This question has been translated from Standard ML to OCaml

(*a*) Describe how lazy lists, which have possibly infinite length, can be implemented in OCaml. Illustrate your answer by presenting a function that accepts one (or more) lazy lists and produces another lazy list. [6 marks]

(*b*) A *lazy binary tree* either is empty or is a branch containing a label and two lazy binary trees, possibly to infinite depth. Present an OCaml type to represent lazy binary trees. [2 marks]

(*c*) Present an OCaml function that produces a lazy binary tree whose labels include all the integers, including the negative integers. [3 marks]

(*d*) Present an OCaml function that accepts a lazy binary tree and produces a lazy list that contains all of the tree's labels. [9 marks]

All OCaml code must be explained clearly.

COMPUTER SCIENCE TRIPOS Part IA – 2008 – Paper 1

6 Foundations of Computer Science (LCP)

This question has been translated from Standard ML to OCaml

A puzzle, or one-person game, can be represented in OCaml by two functions:

- a next-state function, which maps a state to a list of possible next states, and
- a wins function, which returns true if the given state counts as a win.

A simple example is a puzzle that has states consisting of positive integers, a next-state function that maps n to $[n+2, n+5]$, and a "wins" function that returns true if $n = 10$. We can win if we start from $n = 2$ but not from $n = 7$.

(*a*) Code a polymorphic datatype `'a puzzle`, to represent a puzzle by the pair of a next-state function and a wins function. [2 marks]

(*b*) Briefly contrast depth-first search, breadth-first search and iterative deepening as techniques for solving such puzzles. [6 marks]

(*c*) Write a function `depth` that accepts a puzzle, a state and a depth limit. It should use depth-first search to determine whether the puzzle can be solved from the given state within the given depth limit. [6 marks]

(*d*) Write a function `breadth` that accepts a puzzle and a state. It should use breadth-first search to determine whether the puzzle can be solved from the given state. [6 marks]

All code must be explained clearly. You may assume that any necessary OCaml data structures or functions are available.

COMPUTER SCIENCE TRIPOS Part IA – 2009 – Paper 1

1 Foundations of Computer Science (LCP)

This question has been translated from Standard ML to OCaml

(*a*) The polymorphic curried function `delFirst` takes two arguments, a predicate (Boolean-valued function) `p` and a list `xs`. It returns a list identical to `xs` except that the first element satisfying `p` is omitted; if no such element exists, then it raises an exception. Code this function in OCaml. [4 marks]

(*b*) Use the function `delFirst` to express the polymorphic function `delFirstElt`, where `delFirstElt x xs` returns a list identical to `xs` except that it omits the first occurrence of `x`. [2 marks]

(*c*) Carefully explain the polymorphic types of these two functions, paying particular attention to currying and equality. [4 marks]

(*d*) A list `ys` is a *permutation* of another list `xs` if `ys` is obtained by rearranging the elements of `xs`. For example, `[2; 1; 2; 1]` is a permutation of `[2; 2; 1; 1]`. Code an OCaml function to determine whether one list is a permutation of another. [4 marks]

(*e*) A list `ys` is a *generalised permutation* of `xs` if `ys` is obtained by rearranging the elements of `xs`, where one element of `xs` is specially treated: it may appear any number of times (including zero) in `ys`. For example, `[1; 2; 1]` is a generalised permutation of `[1; 2]` but `[1; 2; 2; 1]` is not because two elements (1 and 2) appear the wrong number of times in it. Code an OCaml function to determine whether one list is a generalised permutation of another. [6 marks]

All OCaml code must be explained clearly.

COMPUTER SCIENCE TRIPOS Part IA – 2009 – Paper 1

2 Foundations of Computer Science (LCP)

This question has been translated from Standard ML to OCaml

(*a*) Write brief notes on top–down merge sort, contrasting it with insertion sort. State its worst-case and average-case complexity, with brief justification. (There is no need to present OCaml code.) [5 marks]

(*b*) Write brief notes on preorder, inorder and postorder tree traversal. Present efficient code for one of them and state, with justification, its worst-case complexity. [5 marks]

(*c*) The binary search tree t_1 is *superseded by* t_2 provided every (key, value) entry in t_1 is also present in t_2. Code an OCaml function to determine whether one binary search tree is superseded by another. Express its cost in terms of n_1 and n_2, the numbers of entries in t_1 and t_2, respectively. For full credit, the worst-case cost should be no worse than $O(n_1 + n_2)$. [10 marks]

All code must be explained clearly. You may assume that any necessary OCaml data structures or functions are available.

COMPUTER SCIENCE TRIPOS Part IA – 2010 – Paper 1

1 Foundations of Computer Science (LCP)

This question has been translated from Standard ML to OCaml

(*a*) Give an OCaml datatype declaration suitable for representing lazy lists, possibly of infinite length. [2 marks]

(*b*) Code the OCaml function `interleave`, which takes two lazy lists and generates a lazy list containing each of their elements. [2 marks]

(*c*) Code an OCaml function that applies a given function to every element of a lazy list, returning a lazy list of the results (analogously to the function `map`). [3 marks]

(*d*) Code the OCaml function `iterates` which, given a function f and some value x, generates a lazy list containing all the values of the form $f^n(x)$ (that is, $f(\cdots f(x)\cdots)$ with n applications of f) for $n \geq 0$. [3 marks]

(*e*) Code the OCaml function `iterates2` which, given functions f and g and values x and y, generates a lazy list containing all the values of the form $(f^m(x), g^n(y))$ for m, $n \geq 0$. [10 marks]

All OCaml code must be explained clearly and should be free of needless complexity.

2 Foundations of Computer Science (LCP)

This question has been translated from Standard ML to OCaml

(*a*) Write brief notes on the function defined below:

```
let rec fold_left f e = function
  | [] -> e
  | x::xs -> fold_left f (f e x) xs
```

Illustrate your answer by describing the computations performed by the following two functions:

```
let f x = fold_left (fold_left ( * )) 1 x

let g p zs =
  fold_left
    (fun (x, y) z ->
      if p z then
        (z::x, y)
      else
        (x, z::y))
    ([], []) zs
```

[4 marks]

(*b*) Selection sort is a sorting algorithm that works by repeatedly identifying and setting aside the smallest (or largest) item to be sorted. Implement selection sort in OCaml and describe the efficiency of your solution using O-notation. [4 marks]

(*c*) Code an OCaml function to generate a multiplication table in the form of a list of lists of integers. For example, given the argument 3 it should return `[[1; 2; 3]; [2; 4; 6]; [3; 6; 9]]`. [6 marks]

(*d*) Modify your solution to part (*c*) in order to generate a three-dimensional table containing values x_{ijk} computed by calling a supplied 3-argument curried function f. For example, given the argument 2 it should return `[[[`x_{111}`;` x_{112}`];` `[`x_{121}`;` x_{122}`]];` `[[`x_{211}`;`x_{212}`];` `[`x_{221}`;` x_{222}`]]`. [6 marks]

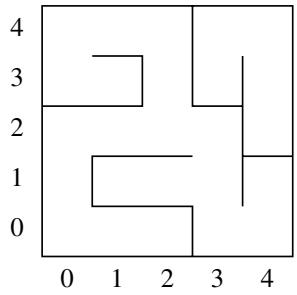
All OCaml code must be explained clearly and should be free of needless complexity.

COMPUTER SCIENCE TRIPOS Part IA – 2011 – Paper 1

1 Foundations of Computer Science (MOM)

This question has been translated from Standard ML to OCaml

This question considers 2-dimensional rectangular labyrinths such as the following.



Here horizontal and vertical bars indicate *walls*. One can *step* from a position to an adjacent position if there is no wall obstructing the move. For example, one can move *in one step* from position (1, 1) to position (2, 1), but not to (1, 0). Diagonal steps are not allowed.

(*a*) Carefully explain a convenient way of representing such labyrinths in OCaml. Then define a function, call it `next`, which takes two arguments, a position `(x, y)` and a labyrinth, and returns a list of all positions that can be reached *in one step* from position `(x, y)`. [Hint: Consider representing labyrinths as functions with a type of the form `int * int ->` *something*.] [8 marks]

(*b*) Use `next` to code, in OCaml, a space-efficient function that checks whether it is possible to move from a position `(x1, y1)` to another position `(x2, y2)` in a given labyrinth. For full marks, make sure your function always terminates. [6 marks]

(*c*) Explain, not necessarily using OCaml code, how one can code a function that returns the *shortest path* between two given positions. Here path means a list of all the positions one would visit en route to the destination. The solution must be space efficient and practical even for large labyrinths. Briefly explain why your solution is space efficient. [6 marks]

2 Foundations of Computer Science (MOM)

This question has been translated from Standard ML to OCaml

(*a*) Write brief notes on exceptions in OCaml and on the functions and control structures available for programming with them. [6 marks]

Parts (*b*) and (*c*) make use of the following OCaml exception:

```
exception Olive
```

(*b*) Code in OCaml a function called `cannot` which takes two arguments, a function `f` and a value `x`. Define the `cannot` function in such a way that it returns `true` if and only if evaluation of `f(x)` causes exception `Olive`. For all other inputs, it should return `false`. [Hint: evaluation of `f(x)` may cause exceptions other than `Olive`.] [4 marks]

(*c*) Consider the following OCaml type and functions `bun` and `cheese`.

```
type 'a tree = Leaf of 'a
             | Branch of 'a tree * 'a tree

let rec bun x = function
  | Leaf y          -> if x = y then raise Olive else Leaf y
  | Branch (t1, t2) -> Branch (bun x t1, bun x t2)

let cheese x t = if cannot (bun x) t then Leaf x else bun x t
```

(*i*) Write down the type of `cheese`. [3 marks]

(*ii*) Write a function that is equivalent to `cheese` but makes no use of exceptions. Briefly explain why your function is equivalent to `cheese`. [7 marks]

COMPUTER SCIENCE TRIPOS Part IA – 2012 – Paper 1

1 Foundations of Computer Science (LCP)

This question has been translated from Standard ML to OCaml

Recall that a dictionary of (*key*, *value*) pairs can be represented by a binary search tree. Define the *union* of two binary search trees to be any binary search tree consisting of every node of the given trees.

(*a*) Write an OCaml function `union` to return the union of two given binary search trees. [*Note:* You may assume that they have no keys in common.] [6 marks]

Define a *slice* of a binary search tree to be a binary search tree containing every (*key*, *value*) node from the original tree such that $x \leq key \leq y$, where x and y are the given endpoints.

(*b*) Write an OCaml function `takeSlice` to return a slice – specified by a given pair of endpoints – from a binary search tree. [4 marks]

(*c*) Write an OCaml function `dropSlice` to *remove* a slice from a binary search tree: given a tree and a pair of endpoints, it should return the binary search tree consisting of precisely the nodes such that $x > key$ or $key > y$. [*Hint:* First consider the simpler task of deleting a node from a binary search tree.] [8 marks]

(*d*) The tree t need not be identical to that returned by

```
union (takeSlice (x, y) t)
      (dropSlice (x, y) t)
```

Briefly explain how such an outcome is possible. [2 marks]

[*Note:* All OCaml code must be explained clearly and should be free of needless complexity.]

2 Foundations of Computer Science (LCP)

This question has been translated from Standard ML to OCaml

(*a*) Write brief notes on `fun`-notation and curried functions in OCaml. Illustrate your answer by presenting the code for a polymorphic curried function `replicate`, which given a non-negative integer n and a value x, returns the list $[\underbrace{x; \ldots; x}_{n}]$. [6 marks]

(*b*) Write brief notes on references in OCaml. Illustrate your answer by discussing (with the aid of a diagram) the effect of the following two top-level declarations:

```
let rlist = replicate 4 (ref 0) @ List.map ref [1; 2; 3; 4]
let slist = List.map (fun r -> ref !r) rlist
```

[6 marks]

(*c*) The following three lines are typed at the OCaml top-level, one after the other. What value is returned in each case? Justify your answer clearly. [*Note:* Recall that an expression of the form v `:=` E has type `unit`.]

```
List.map (fun r -> (r := !r + 1)) rlist
List.map (fun r -> (r := !r - 1; !r)) rlist
List.map (fun r -> (r := !r + 3; !r)) slist
```

[8 marks]

COMPUTER SCIENCE TRIPOS Part IA – 2013 – Paper 1

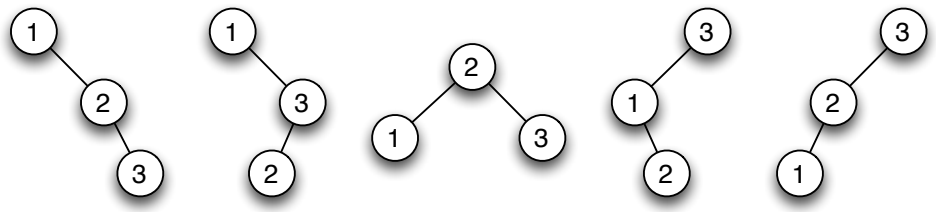
1 Foundations of Computer Science (LCP)

This question has been translated from Standard ML to OCaml

(*a*) Write brief notes on OCaml variants and pattern-matching in function declarations. [6 marks]

(*b*) A binary tree is either a *leaf* (containing no information) or is a *branch* containing a label and two subtrees (called the *left* and *right* subtrees). Write OCaml code for a function that takes a label and two lists of trees, returning all trees that consist of a branch with the given label, with the left subtree taken from the first list of trees and the right subtree taken from the second list of trees. [6 marks]

(*c*) Write OCaml code for a function that, given a list of distinct values, returns a list of all possible binary trees whose labels, enumerated in inorder, match that list. For example, given the list `[1; 2; 3]` your function should return (in any order) the following list of trees:



[8 marks]

All OCaml code must be explained clearly and should be free of needless complexity.

2 Foundations of Computer Science (LCP)

This question has been translated from Standard ML to OCaml

The function `perms` returns all $n!$ permutations of a given n-element list.

```
let rec perms = function
| [] -> [[]]
| xs ->
    let rec perms1 xs ys =
      match xs with
      | [] -> []
      | x::xs ->
          List.map (List.cons x) (perms (List.rev ys @ xs)) @
                   perms1 xs (x::ys)
    in
      perms1 xs []
```

(*a*) Explain the ideas behind this code, including the function `perms1` and the expression `List.map (List.cons x)`. What value is returned by `perms [1; 2; 3]`? [7 marks]

(*b*) A student modifies `perms` to use an OCaml type of lazy lists, where `appendq` and `mapq` are lazy list analogues of `@` and `List.map`.

```
let rec lperms = function
| [] -> Cons ([], fun () -> Nil)
| xs ->
    let rec fun perms1 xs ys = function
    | [] -> Nil
    | x::xs ->
        appendq (mapq (List.cons x) (lperms (List.rev ys @ xs)))
                (perms1 xs (x::ys))
    in
      perms1 xs []
```

Unfortunately, `lperms` computes all $n!$ permutations as soon as it is called. Describe how lazy lists are implemented in OCaml and explain why laziness is not achieved here. [5 marks]

(*c*) Modify the function `lperms`, without changing its type, so that it computes permutations upon demand rather than all at once. [8 marks]

All OCaml code must be explained clearly and should be free of needless complexity.

COMPUTER SCIENCE TRIPOS Part IA – 2014 – Paper 1

1 Foundations of Computer Science (LCP)

This question has been translated from Standard ML to OCaml

(*a*) Write brief notes on polymorphism in OCaml, using lists and standard list functions such as `@` (append) and `List.map`. [4 marks]

(*b*) Explain the meaning of the following declaration and describe the corresponding data structure, including the role of polymorphism.

```
type 'a se = Void | Unit of 'a | Join of 'a se * 'a se
```

[4 marks]

(*c*) Show that OCaml lists can be represented using this variant type by writing the functions `encode_list` of type `'a list -> 'a se` and `decode_list` of type `'a se -> 'a list`, such that `decode_list (encode_list xs) = xs` for every list `xs`. [3 marks]

(*d*) Consider the following function declaration:

```
let rec cute p = function
| Void -> false
| Unit x -> p x
| Join (u, v) ->
    cute p u || cute p v
```

What does this function do, and what is its type? [4 marks]

(*e*) Consider the following expression:

```
fun p -> cute (cute p)
```

What does it mean, and what is its type? Justify your answer carefully. [5 marks]

COMPUTER SCIENCE TRIPOS Part IA – 2014 – Paper 1

2 Foundations of Computer Science (LCP)

This question has been translated from Standard ML to OCaml

(*a*) Write brief notes on the queue data structure and how it can be implemented efficiently in OCaml. In a precise sense, what is the cost of the main queue operations? (It is not required to present OCaml code.) [6 marks]

(*b*) Run-length encoding is a way of compressing a list in which certain elements are repeated many times in a row. For example, a list of the form $[a; a; a; b; a; a]$ is encoded as $[(3, a); (1, b); (2, a)]$. Write a polymorphic function `rl_encode` to perform this encoding. What is the type of `rl_encode`? [6 marks]

(*c*) The simple task of testing whether two lists are equal can be generalised to allow a certain number of errors. We consider three forms of error:

- *element mismatch*, as in [1; 2; 3] versus [1; 9; 3] or [1; 2; 3] versus [0; 2; 3]
- *left deletion*, as in [1; 3] versus [1; 2; 3] or [1; 2] versus [1; 2; 3]
- *right deletion*, as in [1; 2; 3] versus [1; 3] or [1; 2; 3] versus [1; 2]

Write a function `genEquals n xs ys` that returns `true` if the two lists `xs` and `ys` are equal with no more than `n` errors, and otherwise `false`. You may assume that n is a non-negative integer. [8 marks]

All OCaml code must be explained clearly and should be free of needless complexity.

COMPUTER SCIENCE TRIPOS Part IA – 2015 – Paper 1

1 Foundations of Computer Science (LCP)

This question has been translated from Standard ML to OCaml

(*a*) Write brief notes about a tree representation of functional arrays, subscripted by positive integers according to their representation in binary notation. How efficient are the lookup and update operations? [6 marks]

(*b*) Write an OCaml function `arrayoflist` to convert the list $[x_1; \ldots; x_n]$ to the corresponding functional array having x_i at subscript position i for $i = 1, \ldots, n$. Your function should not call the update operation. [6 marks]

(*c*) Consider the task of finding out which elements of an array satisfy the predicate `p`, returning the corresponding subscript positions as a list. For example, the list $[2; 3; 6]$ indicates that these three designated array elements, and no others, satisfy `p`. Write an OCaml functional to do this for a given array and predicate, returning the subscripts in increasing order. [8 marks]

All OCaml code must be explained clearly and should be free of needless complexity.

COMPUTER SCIENCE TRIPOS Part IA – 2015 – Paper 1

2 Foundations of Computer Science (LCP)

This question has been translated from Standard ML to OCaml

(*a*) Write brief notes on programming with lazy lists in OCaml. Your answer should include the definition of a polymorphic type of infinite lazy lists, a function to return the tail of a lazy list, a function to create the infinite list of all positive integers, and an apply-to-all functional analogous to the list functional `map`. [6 marks]

(*b*) Write a function `diag` that takes a lazy list of lazy lists,

$$\begin{array}{l}[\,[z_{11}; z_{12}; z_{13}; \ldots]; \\ \;[z_{21}; z_{22}; z_{23}; \ldots]; \\ \;[z_{31}; z_{32}; z_{33}; \ldots]; \ldots\,]\end{array} \qquad (*)$$

and returns the diagonal, namely the lazy list $[z_{11}; z_{22}; z_{33}; \ldots]$. [3 marks]

(*c*) Write a function that takes two lazy lists $[x_1; x_2; x_3, \ldots]$ and $[y_1; y_2; y_3, \ldots]$ and a function `f` of two arguments; it should return a lazy list of lazy lists like $(*)$ above, with $z_{ij} = \mathtt{f}\, x_i\, y_j$. [3 marks]

(*d*) Write a function that converts a lazy list of lazy lists like $(*)$ above to a lazy list whose elements are all of the z_{ij}, enumerated in some order. [8 marks]

COMPUTER SCIENCE TRIPOS Part IA – 2016 – Paper 1

1 Foundations of Computer Science (LCP)

This question has been translated from Standard ML to OCaml

(*a*) Write brief notes on functions as values and results in OCaml, illustrated with the help of the functionals `map` and `exists`. What functions can we obtain from these via currying? [6 marks]

(*b*) Consider the function `zarg` defined below:

```
let rec zarg f l e = match l with
  | [] -> e
  | x::xs -> f x (zarg f xs e)
```

Show that with the help of this function, it is possible to write an expression for the sum of a given list of integers. Then describe what `zarg` does in general. [4 marks]

(*c*) A polymorphic type of branching trees can be declared as follows. Note that the children of a branch node are given as a *list* of trees, and that only the leaf nodes carry labels.

```
type 'a vtree = Lf of 'a
              | Br of ('a vtree) list
```

(*i*) Write a function `flat t` that converts a given tree `t` of this type to a list of the labels (without eliminating duplicates). Your function should run in linear time in the size of the tree. [4 marks]

(*ii*) Write a function `count x t` that counts the number of times that `x` occurs as a label in `t`, but without first converting `t` to a list.
Note: Minimal credit will be given for solutions that use `flat`. [5 marks]

(*iii*) What is the type of `count`? [1 mark]

All OCaml code must be explained clearly and should be free of needless complexity.

2 Foundations of Computer Science (LCP)

This question has been translated from Standard ML to OCaml

(*a*) A *prime number sieve* is an algorithm for finding all prime numbers up to a given limit n. The algorithm maintains a list, which initially holds the integers from 2 to n. The following step is then repeated: remove the head of this list, which will be a prime number, and remove all its multiples from the list. Write code for the algorithm above as an OCaml function of type `int -> int list`. [4 marks]

(*b*) Consider the problem of eliminating all duplicates from a list of strings. Write code for a function of type `string list -> string list` such that the output contains the same elements as the input, possibly reordered, but where every element occurs exactly once. The worst-case performance must be better than quadratic in the length of the list. [6 marks]

(*c*) Consider the task of determining whether a given *word* (a string) can be expressed by joining together various *chunks* (non-empty strings). If the chunks are *abra*, *cad* and *hal*, then the word *abracadabra* can be expressed as *abra*|*cad*|*abra*. Note that if the available chunks are *ab*, *bra*, *cad* and *abra*, then the first two are no good for expressing *abracadabra*, and yet a solution can be found using *cad* and *abra*. The chunks can be used any number of times and in any order.

Write code for a function that takes a list of chunks along with a word, and returns a list of chunks that yield the word when concatenated. For the examples above, the result should be `["abra"; "cad"; "abra"]`. Exception `Fail` should be raised if no solution exists. [10 marks]

Note: You are given a function `delPrefix` for removing an initial part of a string. For example, `delPrefix "abra" "abracadabra"` returns `"cadabra"`, but `delPrefix "bra" "abracadabra"` raises exception `Fail`.

All OCaml code must be explained clearly and should be free of needless complexity. Well-known utility functions may be assumed to be available.

COMPUTER SCIENCE TRIPOS Part IA – 2017 – Paper 1

1 Foundations of Computer Science (LCP)

This question has been translated from Standard ML to OCaml

A one-person game (such as Rubik's cube, or peg solitaire) has a finite number of possible *states*, some of which count as *winning*. A *move* is a step from one state to another. From each given state, the player can choose from a set of (zero or more) possible next moves. We call a state *winnable* if a winning state can be reached from it in zero or more moves.

For simplicity, assume that states are coded as integers. Also assume that we are given functions `winning x` returning true or false and `next x` returning the list of states that can be reached in one move from state `x`.

(*a*) The following code is an attempt to implement the notion of winnable:

```
let rec exists p = function
  | [] -> false
  | x::xs -> p x || exists p xs
let rec winnable x = winning x || exists winnable (next x)
```

Briefly explain how this code works. Also describe its main limitation: how it can fail to find a winning state that is only a few moves away. Illustrate this point by giving specific definitions of `winning` and `next`. [5 marks]

(*b*) Modify the code above to yield the function `winpath x`, which returns the list of states from `x` to the winning state found or, alternatively, the empty list to indicate that no winning state was found. [4 marks]

(*c*) Sometimes we are only interested in a winnable state if it is only a few moves away from the current state. Modify your solution from part (*b*) to obtain the function `bounded_winpath n x`, which looks for winning states that are at most `n` moves away from `x`. [3 marks]

(*d*) Use your solution from part (*c*) to obtain the function `new_winpath x`, which has the same objective as `winpath x`, but without the limitation mentioned in part (*a*). Briefly explain why the limitation no longer applies and the price that has been paid for this. [5 marks]

(*e*) Briefly outline an alternative approach to correcting the limitation mentioned in part (*a*), using the notion of a queue. What are the advantages and drawbacks of this approach? [3 marks]

For full credit, code should be concise and clear. Exceptions may be useful in this but are not required.

2 Foundations of Computer Science (LCP)

This question has been translated from Standard ML to OCaml

(*a*) Define an OCaml variant type for infinite lists, without the possibility of finite lists. Briefly illustrate programming techniques for your variant type by declaring

(*i*) a recursive functional (analogous to map for ordinary lists) that applies a given function to every element of an infinite list.

(*ii*) a function for generating infinite lists of the form x, $f(x)$, $f(f(x))$, ... for any given f and x.

[6 marks]

(*b*) Briefly explain why the function analogous to append (@) for ordinary lists is of no value for your infinite list data type. Code a function to combine two arbitrary infinite lists, yielding a result that includes the elements of both lists.
[3 marks]

(*c*) Use the functions declared in your previous solutions to express an infinite list consisting of all numbers of the form $5^i \times 7^j \times 9^k$ for integers i, j, $k \geq 0$.
[3 marks]

(*d*) The list `[1; 5; 7; 25; 9; 35; 35; ...]` is a legitimate solution to part (*c*) above, but note that the integers are out of order. Code a function to merge two ordered infinite lists, and use that to modify your previous solution to yield the same set of numbers but in strictly increasing order. Briefly comment, with justification, on whether merge sort for ordinary lists can be generalised to infinite lists.
[8 marks]

For full credit, code should be concise and clear.