Indian Institute of Technology Delhi
Department of Computer Science and Engineering

COL100 — Introduction to Computer Science — Minor II
October 9, 2014 — 9:30–10:30 — Maximum Marks: 60

NOTE: ROUGH SHEETS WILL NOT BE COLLECTED.

Open notes. Write your NAME, ENTRY number and GROUP number at the TOP of each sheet in the blanks provided. Answer all questions in the space provided, in BLUE or BLACK ink (**no pencils, no red pens**). Budget your time according to the marks. Do all rough work on separately provided sheets (not to be submitted).

Question	1 (10 marks)	2 (15 marks)	3 (35 Marks)	Total (60 marks)
Marks	0	7+3	4.5	11.5+3 = 14.5

Q1. (10 marks) A year is a *leap year* if it is a multiple of 4, except if it is a multiple of 100 (not a leap year), unless it is divisible by 400 (is a leap year). Write a program `leapyear: int -> bool`, that responds whether a given year is a leap year or not. # Exception raise Failure ;;

```
let leapyear n = if n < 400 and &
let rec gcd (a, b) = if a>0 or b>0 then
                       if a<b then gcd (b, a)
                       else if a=b then a else
                       gcd (b, a-b) else raise Negative Failure
in if gcd (n, 400) > 1 and n > 400 then true
else if gcd (n,100) > 1 & n>100) then false & gcd (n,400)=1 t
else if gcd (n, 4) > 1 & n > 4 then true
else false ;;
```

(0)

Q2.1 (3 marks) The function `rms_n: float -> float list -> float`, given a `float` n and a list of `floats` $l = [x_1; \ldots; x_m]$, computes the *"root mean square"* of all the elements in l with absolute value strictly greater than n, i.e., it computes $\sqrt{(x_{i_1}^2 + \ldots + x_{i_k}^2)/k}$, where $k \leq m$ is the number of values in l whose absolute value is greater than n. What is `rms_n 2.0 [4.0; -1.0; -3.0; 1.5; 2.0]`? (Note: You can give the answer in the form of $\sqrt{\cdots}$.)

$|x|>n$
$|x|>2$

$\sqrt{\frac{(4)^2+(-3)^2}{2}} = \sqrt{\frac{25}{2}}$ (3)

Q2.2 (12 marks) Write the function `rms_n: float -> float list -> float` using `filter`, `length`, `map` and `foldr`. If your function doesn't these functions then marks will be deducted, even if it works.

(7)

```
let rec rms_n n l = match l with
                    [] -> 0.
                  | (b::bs) ->
```

Sqrt (~~Sqrt~~ mult 1/length) (foldr addfc 0.0 (map square (fill n l)))

Sqrt (mult (1/length l)) (foldr addfc 0.0 (map square (fill n

Helper:- let rec fill n l = match l with
[] -> []
| b::bs -> if b>=n then fill n bs else
b::(fill n bs)

Q3. (8+9+9+9 = 35 marks) **Connectivity in Graphs.** A *graph* is a tuple (V, E) where V is some set of *nodes* and $E \subseteq V \times V$ is a set of *edges* that link nodes. For example, if we consider the cities in India which have airports as the set V, then we can can place edges between all the pairs of cities that have a direct, non-stop, flight between them. One way of representing a graph in Ocaml is by using a square `bool` matrix i.e. a `bool list list`. Each row of the matrix represents one node of the graph and the columns represent the nodes of the set V. If the node i has an edge to the node j then the (i,j)th entry of the matrix is `true`, and so is the (j,i)th entry. *Note: We will assume that a node is trivially connected to itself, i.e., the (i,i)th entry of the matrix is always* `true`. We will use the matrix implementation we studied in class where the empty matrix is represented by an empty list, and the lists within a non-empty list of lists are not allowed to be empty. **NOTE: You may NOT use any of the matrix functions defined in the lectures or notes except where explicitly stated.**

Q3.1 (8 marks) The function `valid_graph: bool list list -> bool` that checks whether a given list of boolean lists is a valid representation of a graph or not can be written as follows:

```
let valid_graph A = (validMatrix A) && (is_square A) && (is_symmetric A) && (is_diagonal A)
```

The function `validMatrix : 'a list list -> bool` was already defined in the notes, so we don't need to define it but we need to define the other 3 functions. You may use the matrix functions `dims: 'a list list -> int * int` to compute the dimensions of a matrix; `transpose: 'a list list -> 'a list list` that computes the transpose of a matrix and `equalMatrix: 'a list list -. 'a list list -> bool` that given two matrices (of the same dimension) returns `true` if they are exactly equal in each entry. Note that you *must* use higher order functions like `foldr`, `map` and `filter` where appropriate. Otherwise you will be penalised even if the program is correct.

```
(* is_square : 'a list list -> bool
   checks if a matrix is square *)
let is_square l = match l with [] -> true
                  | [b]::bs -> true
                  | b::bs -> if let dim l = (m1, n1) and dim (transpose l) = (m2, n2) in if (m1, n1) = (m2, n2) then true else fa

(* is_symmetric: 'a list list -> bool
   checks if (i,j)th entry is equal to (j,i)th entry for all i,j *)
let is_symmetric (l : matrix) = match l with
                  [] -> true | [b] -> true
                  |(b::bs) -> if (equalmatrix l (transpose l)) then true
                  else false;;

(* is_diagonal : bool list list -> bool
   checks if all the entries along the diagonal are true *)
let is_diagonal l = match l with [] -> true
                  | [b] -> true
                  | b::bs -> if validmatrix l then if
                  (is_symmetric l) && (is_square l) &&
                  map (filteri zero l) = nth(0,1)
                  then true else false.
```

Name: Shalinee kumari meena Entry: 2014ME10684 Gp: 1

Q3.2 (9 marks) Given a `bool list list` A that represents a graph (V, E), we can define a boolean matrix A^2 in which the (i, j)th and (j, i)th entries are true if there is a path that connects nodes i and j with *at most one* intermediate node. Write a function `square_graph: bool list list -> bool list list` that given a graph A, computes A^2 which gives the connectivity of nodes by paths of length up to 2 in that graph.

let rec square_graph A = matrix A with [] -> true

Q3.3 (9 marks) Given a `bool list list` A that represents a graph (V, E) with n nodes and e edges, we define A^*, called the *transitive closure* of A, as the boolean matrix in which the (i, j)th and (j, i)th entries are true if there is a path (with any number of intermediate nodes) that connects nodes i and j. Write a function `transitive_closure : bool list list -> bool list list` that computes the transitive close of a graph.

Name: Shelinee kumari meena Entry: 2014ME10684 Gp: 1

Q3.4 (9 marks) Two nodes i, j of a graph (V, E) are said to be *connected* if there is a path with any number of intermediate nodes that connects them in the graph. Connectedness is an equivalence relation, and its equivalence classes are called *connected components*. Write a function `conn_components : bool list list -> int list list` that takes a graph as input and returns an `int list list` in which each inner list contains all the nodes (represented as integers) of a connected component.

(Spare space if required)