3. datatype definition for a type number whose value are either integers or real numbers.  
  
datatype number = Int of int

| Real of real;

4. using your definition from exercise 3, write a function plus of type number -> number -> number that adds two numbers. Coercing int to real only if necessary.  
  
fun plus (Int i1, Int i2) = Int (i1 + i2)

| plus (Int i, Real r) = Real ((real i) + r)

| plus (Real r, Int i) = Real (r + (real i))

| plus (Real r1, Real r2) = Real (r1 + r2);

5.write a function addup of type intnest -> int that adds up all the integers in an intnest. User this definition for intnest. (becareful as you type. INT is not the same as int)  
  
 datatype intnest =   
 INT of int |  
 LIST of intnest list;  
  
fun addup (INT i) = i

| addup (LIST xs) = foldl (fn (x, acc) => acc + addup x) 0 xs;

9. write a function appendall of type ‘a list tree -> ‘a list that takes a tree of lists and returns the result of appending all the lists together. Put the list for a node together in this order” first the contents of the left subtree. Then the list at this node, and then the contents of the right subtree. Here again is the definition of tree, as seen earlier in this chapter:  
 datatype ‘data tree =  
 Empty |  
 Node of ‘data tree \* ‘data \* ‘data tree;

the following input should give the output indicated below:

- val tree = Node(Node(Empty,[1,2],Empty),[3,4],Node(Empty,[5,6],Empty));  
val tree = Node (Node (Empty,[#,#],Empty),[3,4],Node (Empty,[#,#],Empty)): int list tree  
- appendall tree;  
val it = [1,2,3,4,5,6] : int list

fun appendall Empty = []

| appendall (Node (left, data, right)) =

appendall left @ data @ appendall right;  
  
  
10. a complete binary tree is one in which every Node has either two Empty children or two Node children, but not one of each. Write a function is Complete of type ‘a tree -> bool that tests whether a tree is complete. (use the tree definition from Exercise 9.)  
following should occur:

- val tree1 = Node(Node(Empty,1,Empty),2,Node(Empty,3,Empty));  
val tree1 = Node (Node (Empty,1,Empty),2,Node (Empty,3,Empty)) : int tree  
- val tree2 = Node(Node(Empty,1,Empty),2,Empty);  
val tree2 = Node (Node (Empty,1,Empty),2,Empty) : int tree  
- isComplete tree1;  
val it = true : bool  
- isComplete tree2;  
val it = false : bool  
- isComplete Empty;   (\* An Empty tree is vacuously complete \*)  
val it = true : bool

a.   
datatype entry = File of string | Folder of string \* entry list;

fun print\_entries (File fname) = print (fname ^ "\n")

| print\_entries (Folder (dname, contents)) =

(print (dname ^ "\n"); print\_contents contents)

and print\_contents L = List.app print\_entries L;

b.

the function f takes a function g and a list and applies g to each list element with g being applied an additional time for each element