signature MATHLIB = sig
  val fact : int → int
  val half_pi : real
  val doubler : int → int
end

structure MyMathLib => MATHLIB = struct
  fun fact x = 
    if x<0 then 1 
    else x × fact (x - 1)
  val half_pi = Math.pi / 2.0
  fun doubler y = y + y
end

val pi = MyMathLib.half_pi + MyMathLib.half_pi
val twenty_eight = MyMathLib.doubler 14

and print_rat may print a non-reduced fraction.
We fix this by making rational abstract. *)

signature RATIONAL_B = sig
  type rational (* type now abstract *)
  exception BadFrac
  val make_frac : int X int ⇒ rational
  val add : rational X rational ⇒ rational
  val toString : rational ⇒ string
end

(* as a cute trick, it is actually okay to expose
the Whole function since no value breaks
our invariants, and different implementations
can still implement Whole differently. *)

signature RATIONAL_C = sig
  type rational (* type still abstract *)
  exception BadFrac
  val make_frac : int X int ⇒ rational
  val add : rational X rational ⇒ rational
  val toString : rational ⇒ string
end

(* can ascribe any of the 3 signatures above *)

structure Rational1 = struct
  datatype rational = Whole of int
                  | Frac of int × int
  exception BadFrac
  fun gcd (x,y) = 
    if x=y then x 
    else if x < y then gcd(x,y−x) 
    else gcd(y,x)
  fun reduce r = 
    case r of
      Whole _ ⇒ r
    | Frac(x,y) ⇒ 
      if x=0 then Whole 0
      else if y < 0 then reduce(Frac(~x,~y))
      else reduce(Frac(x,y))
  fun make_frac (x,y) = 
    if y = 0 then raise BadFrac
    else if y < 0 then reduce(Frac(-x,~y))
    else reduce(Frac(x,y))
fun add (r1,r2) =  
  case (r1,r2) of  
    (Whole(i),Whole(j)) => Whole(i+j)  
  | (Whole(i),Frac(j,k)) => Frac(j+i*k,k)  
  | (Frac(j,k),Whole(i)) => Frac(j+k*i,k)  
  | (Frac(a,b),Frac(c,d)) => reduce (Frac(a*c + b*d, b*d))  

(* given invariant, prints in reduced form *)  
fun toString r =  
  case r of  
    Whole i => Int.toString i  
  | Frac(a,b) => (Int.toString a) ^ "/" ^ (Int.toString b)  
end  

(* this structure can have all three signatures we gave Rational, and/or it is /equivalent/ under signatures RATIONAL_B and RATIONAL_C  
this structure does not reduce fractions until printing *)  
structure Rational2 : RATIONAL_A (* or B or C *) =  
  struct  
    datatype rational = Whole of int | Frac of int*int  
    exception BadFrac  
    fun make_frac (x,y) =  
      if y = 0 then raise BadFrac  
      else if y < 0 then Frac(~x,~y)  
      else (x,y)  
    fun whole i = (i, 1)  
    fun add ((a,b),(c,d)) = (a*d + c*b, b*d)  
    fun toString r = let  
      fun gcd (x,y) =  
        if x = y then x  
        else if x < y then gcd(x,y-x)  
        else gcd(y,x)  
      val d = gcd(abs x,y)  
      val num = x div d  
      val denom = y div d  
      in  
        Int.toString num ^ (if denom=1 then "" else "/" ^ (Int.toString denom))  
      end  
  end  

(* this structure uses a different abstract type. It does not even have signature RATIONAL_A. For RATIONAL_C, we need a function Whole. *)