fun compose (f, g) = λ x ⇒ f (g x)
fun double x = x + x
fun incr x = x + 1

val _ = print (Int.toString ((compose (double, incr)) 5))

(* SML provides compose as infix op "o" *)

val _ = print (Int.toString ((double ° incr) 5))

fun sqrt_of_abs i = Math.sqrt (Real.fromInt (abs i))

fun sqrt_of_abs i = (Math.sqrt ° Real.fromInt ° abs) i

val sqrt_of_abs = Math.sqrt ° Real.fromInt ° abs

val x = sqrt_of_abs ~49

infix 3 <\ fun x <\ f = λ y ⇒ f (x, y) (* Left section *)
infix 3 \> fun f \> y = f y (* Left application *)
infixr 3 */ fun f */ y = λ x ⇒ f (x, y) (* Right section *)
infixr 3 </ fun x </ f = f x (* Right application *)
infix 2 ° (* See motivation below *)
infix 0 :=
infix 1 |> fun op |> y = op y (* Left pipe *)
infixr 1 <| fun op <| y = op y (* Right pipe *)

(* currying *)

fun flip f y x = f x y
fun strApp x y = x ^ y

val _ = incr > double > Int.toString > "\n\n" * op * print

(* *)

val _ = 5 > incr > double > Int.toString ° "\n\n" * "\n\n" < op > print

fun filter f [] = []
    | filter f (x :: xs) = if f x
    then x :: filter f xs
    else filter f xs

val remNegs = filter (λ x ⇒ x ≥ 0)

fun fold' f acc [] = acc
    | fold' f acc (x :: xs) = fold' f (f x acc) xs

fun tuple_sorted (x, y, z) = x ≤ y ∧ y ≤ z
fun cmpA x y z = x ≤ y ∧ y ≤ z

val t1 = tuple_sorted (1, 2, 3)
val t2 = ((cmpA 1) 2) 3
val cmpB = cmpA l