|  |  |
| --- | --- |
| **CHURCH ENCODING:**  type 'a church = Church of (('a → 'a) → 'a → 'a)  let zero = Church (fun f x → x)  let one = Church (fun f x → f x)  let two = Church (fun f x → f (f x))  let **iszero** (Church c) = (c fun x → false)) true  let rec **create** (n : int) : 'a church =  match n with  | 0 → Church (fun f x → x)  | \_ → match create (n - 1) with  **(\*** Add 1 to the Church numeral representing n - 1 **\*)**  | Church c → Church (fun f x → f (c f x))  let **churchToInt** (Church c) = c (fun n → n + 1) 0  let **succ** (Church c) = Church (fun f x → f (c f x))  let **add** (Church cm) (Church cn) = Church (fun f x → cm f (cn f x))  let **mult** (Church cm) (Church cn) = Church (fun f x → cn (cm f) x)  let **exp** (Church cm) (Church cn) = Church (cn cm)  **HOF:**  **(\* curry** : ('a → 'b → 'c) → (('a \* 'b) → 'c) **\*)**  let **curry** f = (fun x y → f (x,y))  **{\*** **uncurry** : ((‘a \* ‘b) → ‘c) → (‘a → ‘b → ‘c) **\*)**  let **uncurry** f = (fun (y, x) → f y x)  let **swap** f = (fun (x,y) → f (y,x))  let rec **exp** (b,n) = if n = 0 then 1 else b \* exp(b, n-1)  let rec **sumSquare** (a,b) = if (a > b) then 0 else square(a) + sumSquare(a+1,b)    let **compute\_all\_results** l c = List.map (function  (id, amt, Some b) → let win = compute b c in (id, amt + win, None )  | (id, amt, None) → (id, amt, None)) l  let **compute\_winners** l c = **List.filter** (function (id, amt, Some b) → compute b c > 0 | (id, amt, None) → false) l  **FOLDING:**  **List.fold\_right** (fun h acc → h + acc) [1;2;3;4] 0;; **// 10**  **List.fold\_right** (fun h acc → (string\_of\_int h) ^ acc) [1;2;3;4] “”;; **// “1234”**  **List.fold\_right** (fun h acc → if h mod 2 = 0 then h + acc else acc) [1;2;3;4] 0;; **// sum of even nums**  **List.fold\_left** (fun acc h → (string\_of\_int h) ^ acc) “” [1;2;3;4];; **// “4321”** | **TREES:**  type 'a tree = Empty | Node of 'a \* 'a tree \* 'a tree  let rec **insert** ((x,d) as e) t = match t with  | Empty → Node(e, Empty, Empty)  | Node ((y,d'), l, r) →  if x = y then Node(e, l, r)  else  (if x < y then  Node((y,d'), insert e l, r)  else  Node((y,d'), l, insert e r))  let rec **lookup** x t = match t with  | Empty → None  | Node ((y,d), l, r) →  if x = y then Some(d)  else  (if x < y then lookup x l  else lookup x r)  **LISTS:**  [None; Some 3; Some 5; None] **// int option list**  **DATA TYPES:**  type ‘a option = None | Some of ‘a  **OTHERS:**  let rec **generate\_deck** suits ranks =  match suits, ranks with  | [], \_ | \_, [] → []  | suit\_h::suit\_t, rank\_h::rank\_t →  (rank\_h, suit\_h)::  (generate\_deck [suit\_h] rank\_t)@  (generate\_deck suit\_t ranks)  **(\*** [Hearts; Spades] [Seven; Nine; Ten; Ten; Eight] **gives**  [(Seven, Hearts); (Nine, Hearts); (Ten, Hearts); (Ten, Hearts); (Eight, Hearts); (Seven, Spades); (Nine, Spades); (Ten, Spades); (Ten, Spades); (Eight, Spades)] **\*)** |