fun identity (x) = x

(* CSE 341: Section 4   Topics in this file:   −> anonymous functions   −> map, flat_map, fold   −>*)(* Here’s a pretty boring function *)(* it’s going to have type ‘a −> ‘a *)(* it takes anything, and returns that same type *)

⇒

|x|::xs'⇒ if p x
|   |   ⇒ filter(p, xs');

full_typed_identity (x : α) : α = x

(* You’ve seen type variables in types *)(* you can write them yourself *)(* this is exactly the same as above *)

datatype α Option = Some of α | None

fun map (f, xs) =

(* map! what if we want to apply some function over a list and return   the resulting list? *)

|  x|::xs'⇒ (f x) ⇒ ::(map (f, xs'))

fun double_all xs =
map (x λ x ⇒ 2, xs)

fun double x =
(* anonymous functions *)(* we’ve seen function bindings like the following *)

x 2;

val doubled = double_all [1, 2, 3, 4, 5]

val double' =
(* we can rewrite this: *)

λ x x ⇒ 2;

fun replicate x =
(* what about functions that return functions? *)(* Disclaimer: Only these three because it is a small example and the character   sets are "similar enough" to ascii. *)

datatype Lang = Spanish | French | English | German

fun add_greeting Spanish = (λ str ⇒ "Hola " ^ str)

| add_greeting French = (λ str ⇒ "Bonjour " ^ str)

| add_greeting English = (λ str ⇒ "Hello " ^ str)

| add_greeting German = (λ str ⇒ "Hallo " ^ str)

val es_greet = add_greeting Spanish

val fr_greet = add_greeting French

val en_greet = add_greeting English

val de_greet = add_greeting German

val str1 = es_greet "Mundo"

val str2 = fr_greet "Monde"

val str3 = en_greet "World"

val str4 = de_greet "Welt"

fun filter(p, xs) =

| case xs of

| [] ⇒ []

| x::xs'⇒ if p x

|   then x :: filter(p, xs')

| else filter(p, xs');

(* map! what if we want to apply some function over a list and return   the resulting list? *)

fun map (f, xs) =

| case xs of

| [] ⇒ []

| x::xs'⇒ (f x) :: map (f, xs')

fun double_all xs =
map (λ x ⇒ x x 2, xs)

val doubled = double_all [1, 2, 3, 4, 5]

fun replicate x =

let

| fun rep_helper (x, copies) =

|   if copies = 0

|   then []

|   else x :: rep_helper (x, copies − 1)

| in

| rep_helper (x, x)

| end

fun make_replicates xs =
map (replicate, xs)

(* what if we want to flatten it? *)

fun flat_map (f, xs) =

case xs of

| [] ⇒ []

| x::xs'⇒ f(x) @ flat_map (f, xs')

fun make_flat_replicates xs =

| flat_map (replicate, xs)

(* fold! (see also lecture) *)

fun fold (f,acc,xs) =

| case xs of

| [] ⇒ acc

| x::xs'⇒ fold (f,f(acc,x),xs')

fun count_zeros xs = fold((λ (acc,x) ⇒ if x=0 then acc+1 else acc), 0, xs)

type date = int × int × int

fun day (d : date) = #1 d

fun month (d : date) = #2 d

fun year (d : date) = #3 d

(* homework 1 revisited *)(* 2 *)(* count how many dates in a list are in the given month *)

fun is_in_month((_,m,_), month) = (m = month);

fun number_in_month(dates, month) =

let

| fun check_date d = is_in_month(d, month)

| in

| length(filter(check_date, dates))

| end;

(* 3 *)(* count how many dates in a list are in the given list of months *)

fun number_in_months(dates, months) =
let
  fun get_month_count m = number_in_month(dates, m)
  in
  sum(map(get_month_count, months))
  end;

(* 5 *)
(* return list of dates that are in given list of months *)
(* could be done with dates_in_month and flat_map *)

(* 6 *)
fun n_times (f,n,x) = 
  if n=0
    then x
    else f (n_times(f,n-1,x))

fun get_nth(xs, n) = hd (n_times(tl, n-1, xs));