```
(* Steve Chapman *)

(* CSC 330 *)
(* Assignment #1 *)
(* January 16, 2015 *)

type DATE = {year:int, month:int, day: int}
exception InvalidParameter

(* This file is where your solutions go *)

fun is_older (d1: DATE, d2: DATE) =
    if (#year(d1) > #year(d2)) orelse ((#year(d1) = #year(d2)) andalso (#month(d1)
>= #month(d2)) andalso (#day(d1) > #day(d2)))
    then true
    else false

fun number_in_month (dl: DATE list, m:int) =
    if null dl
    then 0
    else (if #month(hd(dl))=m
      then 1 + number_in_month(tl(dl), m)
      else number_in_month(tl(dl), m))

fun number_in_months (dl2: DATE list, il2: int list) =
    if null il2
    then 0
    else number_in_month(dl2, hd(il2)) + number_in_months(dl2, tl(il2))

fun dates_in_month (dl3: DATE list, m: int) =
    if null dl3
    then []
    else (if #month(hd(dl3)) = m
      then hd(dl3) :: dates_in_month(tl(dl3),m)
      else dates_in_month(tl(dl3),m))

fun dates_in_months (dl4: DATE list, il4: int list) =
    if null il4
    then []
    else dates_in_month(dl4, hd(il4)) @ dates_in_months(dl4, tl(il4))

fun get_nth (sl: string list, n: int) =
    if n < 1 orelse (n>0 andalso sl = []) then raise InvalidParameter
    else if n=1
    then hd(sl)
    else get_nth(tl(sl), n-1)

fun date_to_string (d: DATE) =
    let val months: string list = ["January", "February", "March", "April", "May",
"June", "July", "August", "September", "October", "November", "December"]
```

```
    in
    get_nth(months, #month(d)) ^ " " ^ Int.toString(#day(d)) ^ ", " ^
Int.toString(#year(d))
    end

fun number_before_reaching_sum (sum: int, il5: int list) =
    if sum > 0 andalso il5 = [] then raise InvalidParameter
    else if sum <= hd(il5)
    then 0
    else 1 + number_before_reaching_sum (sum - hd(il5), tl(il5))

fun what_month (julian_day: int) =
    (* Intentionally using 13 values here to avoid adding an offset of +1 month in
the "in" clauses *)
    let val cumulative_days: int list = [0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31,
30, 31]
    in
    number_before_reaching_sum (julian_day, cumulative_days)
    end

fun month_range (day1: int, day2: int) =
    if day1 > day2
    then []
    else what_month(day1) :: month_range(day1 + 1, day2)

fun oldest (dl11: DATE list) =
    if null dl11
    then NONE
    else
    let val tl_oldest = oldest(tl dl11)
    in
        if isSome tl_oldest andalso is_older(valOf tl_oldest, hd dl11)
        then tl_oldest
        else SOME (hd dl11)
    end

fun reasonable_date (d12: DATE) =
    (* First, check year and month *)
    if #year(d12) <= 0 orelse (#month(d12) < 1 orelse #month(d12) > 12)
    then false
    (* Next, check day - check leap year, then check date *)
    else
    let val month_days: int list = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
        val month_days_leap_year: int list = [31, 29, 31, 30, 31, 30, 31, 31, 30,
31, 30, 31]
        fun get_nth_int (il: int list, n: int) =
        if n < 1 orelse (n>0 andalso il = []) then raise InvalidParameter
        else if n=1
        then hd(il)
        else get_nth_int (tl(il), n-1)
        fun check_month_leap_year () =
        if #day(d12) <= get_nth_int(month_days_leap_year, #month(d12))
        then true
        else false
        fun check_month_non_leap_year () =
        if #day(d12) <= get_nth_int(month_days, #month(d12))
        then true
        else false
        fun check_month () =
        if #year(d12) mod 400 = 0 orelse (#year(d12) mod 4 = 0 andalso #year(d12)
mod 100 <> 0)
        then check_month_leap_year ()
        else check_month_non_leap_year()
    in
        check_month()
    end
```