Міністерство освіти і науки України
Національний технічний університет України «КПІ ім. Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Мультипарадигменне програмування

ЗВІТ

до лабораторних робіт

Виконав студент ІП-01 Ніколаєв Іван Романович _____
(№ групи, прізвище, ім'я, по батькові)

Прийняв ас. Очеретяний О. К. _____
(посада, прізвище, ім'я, по батькові)

Київ 2021

Завдання

Ви напишете 11 функцій SML (і тести для них), пов’язаних з календарними датами. У всіх завданнях, **“дата”** є значенням SML типу int*int*int, де перша частина - це рік, друга частина - місяць і третя частина - день. **«Правильна»** дата має позитивний рік, місяць від 1 до 12 і день не більше 31 (або 28, 30 - залежно від місяця). Перевіряти “правильність” дати не обов'язково, адже це досить складна задача, тож будьте готові до того, що багато ваших функцій будуть працювати корректно для деяких/всіх **“неправильних”** дат у тому числі. Також, **«День року»** — це число від 1 до 365 де, наприклад, 33 означає 2 лютого. (Ми ігноруємо високосні роки, за винятком однієї задачі.)

1. Напишіть функцію is_older, яка приймає дві дати та повертає значення true або false. Оцінюється як true, якщо перший аргумент - це дата, яка раніша за другий аргумент. (Якщо дві дати однакові, результат хибний.)
2. Напишіть функцію number_in_month, яка приймає список дат і місяць (тобто int) і повертає скільки дат у списку в даному місяці.
3. Напишіть функцію number_in_months, яка приймає список дат і список місяців (тобто список int) і повертає кількість дат у списку дат, які знаходяться в будь-якому з місяців у списку місяців. **Припустимо, що в списку місяців немає повторюваних номерів**. Підказка: скористайтеся відповіддю до попередньої задачі.
4. Напишіть функцію dates_in_month, яка приймає список дат і число місяця (тобто int) і повертає список, що містить дати з аргументу “список дат”, які знаходяться в переданому місяці. Повернутий список повинен містять дати в тому порядку, в якому вони були надані спочатку.
5. Напишіть функцію dates_in_months, яка приймає список дат і список місяців (тобто список int) і повертає список, що містить дати зі списку аргументів дат, які знаходяться в будь-якому з місяців у списку місяців. Для простоти, припустимо, що в списку місяців немає повторюваних

номерів. Підказка: Використовуйте свою відповідь на попередню задачу та оператор додавання списку SML (@).

6. Напишіть функцію get_nth, яка приймає список рядків і int n та повертає n-й елемент списку, де голова списку є першим значенням. Не турбуйтеся якщо в списку занадто мало елементів: у цьому випадку ваша функція може навіть застосувати hd або tl до порожнього списку, і це нормально.
7. Напишіть функцію date_to_string, яка приймає дату і повертає рядок у вигляді “February 28, 2022” Використовуйте оператор ^ для конкатенації рядків і бібліотечну функцію Int.toString для перетворення int в рядок. Для створення частини з місяцем не використовуйте купу розгалужень. Замість цього використайте список із 12 рядків і свою відповідь на попередню задачу. Для консистенції пишіть кому після дня та використовуйте назви місяців англійською мовою з великої літери.
8. Напишіть функцію number_before_reaching_sum, яка приймає додатний int під назвою sum, та список int, усі числа якої також додатні. Функція повертає int. Ви повинні повернути значення int n таке, щоб перші n елементів списку в сумі будуть менші sum, але сума значень від n + 1 елемента списку до кінця був більше або рівний sum.
9. Напишіть функцію what_month, яка приймає день року (тобто int між 1 і 365) і повертає в якому місяці цей день (1 для січня, 2 для лютого тощо). Використовуйте список, що містить 12 цілих чисел і вашу відповідь на попередню задачу.
10. Напишіть функцію month_range, яка приймає два дні року day1 і day2 і повертає список int [m1,m2,...,mn] де m1 – місяць day1, m2 – місяць day1+1, ..., a mn – місяць day2. Зверніть увагу, що результат матиме довжину day2 - day1 + 1 або довжину 0, якщо day1>day2.
11. Напишіть найстарішу функцію, яка бере список дат і оцінює параметр (int*int*int). Він має оцінюватися як NONE, якщо список не містить дат, і SOME d, якщо дата d є найстарішою датою у списку.

Програмний код

(task.sml)

```
(*----------------1---------------*)
fun is_older (date1: int* int* int, date2: int* int* int) =
    if (#1 date1) < (#1 date2) then
        true
    else if (#1 date1) = (#1 date2) then
        if (#2 date1) < (#2 date2) then
            true
        else if ((#2 date1) = (#2 date2)) andalso ((#3 date1) < (#3 date2)) then
            true
        else
            false
    else
        false

(*----------------2---------------*)

fun number_in_month (datelist: (int* int* int) list, month: int) =
    if null datelist then
        0
    else
        if #2 (hd datelist) = month then
            1 + number_in_month(tl datelist, month)
        else
            number_in_month(tl datelist, month)

(*----------------3---------------*)

fun number_in_months (datelist: (int* int* int) list, monthlist: int list ) =
    if null monthlist then
        0
    else
        number_in_month(datelist, hd monthlist) + number_in_months(datelist, tl
monthlist)


(*----------------4---------------*)

fun dates_in_month (datelist: (int* int* int) list, month: int) =
    if null datelist then
        []
    else
        if #2 (hd datelist) = month then
            (hd datelist) :: dates_in_month(tl datelist, month)
        else
            dates_in_month(tl datelist, month)
```

```
(*----------------5---------------*)


fun dates_in_months (datelist: (int* int* int) list, monthlist: int list) =
    if null monthlist then
        []
    else
        dates_in_month(datelist, hd monthlist) @ dates_in_months(datelist, tl
monthlist)


(*----------------6---------------*)

fun get_nth (stringlist : string list, n : int) =
    if null stringlist then
        ""
    else
        if n = 1 then
            hd stringlist
        else
            gen_nth(tl stringlist, n - 1)


(*----------------7---------------*)

fun date_to_string (date: int* int* int) =
    gen_nth(["January", "February", "March", "April", "May", "June", "July",
"August", "September", "October", "November", "December"]
    , #2 date) ^ " " ^ (Int.toString (#3 date)) ^ ", " ^ (Int.toString (#1 date))


(*----------------8---------------*)

fun number_before_reaching_sum (sum: int, intlist: int list) =
    if null intlist then
        0
    else
        if (sum - (hd intlist)) <= 0 then
            0
        else
            1 + number_before_reaching_sum(sum - (hd intlist), tl intlist)


(*----------------9---------------*)
fun what_month (day: int) =
    number_before_reaching_sum(day, [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30,
31]) + 1


(*----------------10---------------*)
fun month_range (day1: int, day2: int) =
    if day1 > day2 then
```

```
        []
    else
        what_month(day1) :: month_range(day1 + 1, day2)


(*-----------------11----------------*)
fun oldest_date (datelist: (int* int* int) list) =
    if null datelist then
        NONE
    else
        let fun oldest_date_nonempty (datelist : (int*int*int) list) =
                if null (tl datelist) then
                    hd datelist
                else
                    let val tl_ans = oldest_date_nonempty(tl datelist)
                    in
                        if is_older(hd datelist, tl_ans) then
                            hd datelist
                        else
                            tl_ans
                    end
        in
            SOME (oldest_date_nonempty(datelist))
        end;
```

Тести

(test.sml)

```
use "task.sml";

fun test(function_name : string, true_result, fact_result) =
    if true_result = fact_result
    then (function_name, "Passed")
    else (function_name, "Failed");

(* 1 *)
test("is_older", true, is_older((2021, 5, 30), (2021, 5, 31)));
test("is_older", false, is_older((2022, 5, 5), (2021, 10, 21)));
test("is_older", false, is_older((2003, 5, 5), (2003, 5, 5)));


(* 2 *)
test("number_in_month", 3, number_in_month([(2021, 5, 30), (2021, 5, 30), (2021,
5, 30), (2021, 6, 30), (2021, 1, 30), (2021, 2, 30)], 5));
test("number_in_month", 1, number_in_month([(2021, 5, 30), (2021, 5, 30), (2021,
5, 30), (2021, 6, 30), (2021, 1, 30), (2021, 2, 30)], 6));
```

```
test("number_in_month", 0, number_in_month([(2021, 5, 30), (2021, 5, 30), (2021,
5, 30), (2021, 6, 30), (2021, 1, 30), (2021, 2, 30)], 4));


(* 3 *)
test("number_in_months", 4, number_in_months([(2021, 5, 30), (2021, 5, 30),
(2021, 5, 30), (2021, 6, 30), (2021, 1, 30), (2021, 2, 30)], [1, 5]));
test("number_in_months", 2, number_in_months([(2021, 5, 30), (2021, 5, 30),
(2021, 5, 30), (2021, 6, 30), (2021, 1, 30), (2021, 2, 30)], [2, 6]));
test("number_in_months", 0, number_in_months([(2021, 5, 30), (2021, 5, 30),
(2021, 5, 30), (2021, 6, 30), (2021, 1, 30), (2021, 2, 30)], []));


(* 4 *)
test("dates_in_month", [(2001, 1, 30), (2002, 1, 30), (2004, 1, 30)],
dates_in_month([(2001, 1, 30), (2002, 1, 30), (2003, 2, 30), (2004, 1, 30),
(2005, 2, 30), (2006, 3, 30)], 1));
test("dates_in_month", [], dates_in_month([(2001, 1, 30), (2002, 1, 30), (2003,
2, 30), (2004, 1, 30), (2005, 2, 30), (2006, 3, 30)], 5));
test("dates_in_month", [(2006, 3, 30)], dates_in_month([(2001, 1, 30), (2002, 1,
30), (2003, 2, 30), (2004, 1, 30), (2005, 2, 30), (2006, 3, 30)], 3));


(* 5 *)
test("dates_in_months", [(2001, 1, 30), (2002, 1, 30), (2004, 1, 30), (2006, 3,
30)], dates_in_months([(2001, 1, 30), (2002, 1, 30), (2003, 2, 30), (2004, 1,
30), (2005, 2, 30), (2006, 3, 30)], [1, 3, 8]));
test("dates_in_months", [], dates_in_months([(2001, 1, 30), (2002, 1, 30), (2003,
2, 30), (2004, 1, 30), (2005, 2, 30), (2006, 3, 30)], [4, 5]));
test("dates_in_months", [], dates_in_months([(2001, 1, 30), (2002, 1, 30), (2003,
2, 30), (2004, 1, 30), (2005, 2, 30), (2006, 3, 30)], []));


(* 6 *)
test("get_nth", "second", get_nth(["first", "second", "third", "fourth",
"fifth"], 2));
test("get_nth", "fifth", get_nth(["first", "second", "third", "fourth", "fifth"],
5));
test("get_nth", "third", get_nth(["first", "second", "third", "fourth", "fifth"],
3));


(* 7 *)
test("date_to_string", "May 5, 2022", date_to_string((2022, 5, 5)));
test("date_to_string", "May 6, 2003", date_to_string((2003, 5, 6)));
test("date_to_string", "November 11, 2011", date_to_string((2011, 11, 11)));


(* 8 *)
test("number_before_reaching_sum", 3, number_before_reaching_sum(7, [1, 1, 1, 4,
5]));
```

```
test("number_before_reaching_sum", 3, number_before_reaching_sum(6, [1, 2, 2, 3, 5]));
test("number_before_reaching_sum", 0, number_before_reaching_sum(4, [10, 2, 3, 4, 5]));


(* 9 *)
test("what_month", 8, what_month(215));
test("what_month", 1, what_month(1));
test("what_month", 1, what_month(31));


(* 10 *)
test("month_range", [], month_range(30, 1));
test("month_range", [1, 2], month_range(31, 32));
test("month_range", [3, 3, 3], month_range(70, 72));


(* 11 *)
test("month_range", NONE, oldest_date([]));
test("month_range", SOME (2001, 1, 29), oldest_date([(2001, 1, 30), (2002, 1, 30), (2001, 2, 30), (2004, 1, 30), (2001, 1, 29), (2006, 3, 30)]));
test("month_range", SOME (2015, 1, 31), oldest_date([(2015, 2, 31), (2015, 2, 31), (2015, 1, 31)]));
```