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 Шпилька Владислав Сергійович |  |  |
|  |  | (№ 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, ..., а 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.

**Реалізація:**

(\* 1 \*)

fun is\_older (first\_date : int\*int\*int, second\_date : int\*int\*int) =

    if #1 first\_date \* 365 + #2 first\_date \* 31 + #3 first\_date < #1 second\_date \* 365 + #2 second\_date \* 31 + #3 second\_date

    then true

    else false;

(\* 2 \*)

fun numbers\_in\_month (dates : (int\*int\*int) list, month : int) =

    if null dates

    then 0

    else (if #2 (hd dates) = month then 1 else 0) + numbers\_in\_month(tl dates, month);

(\* 3 \*)

fun numbers\_in\_months (dates : (int\*int\*int) list, months : int list) =

    if null months

    then 0

    else numbers\_in\_month(dates, hd months) + numbers\_in\_months(dates, tl months);

(\* 4 \*)

fun dates\_in\_month (dates : (int\*int\*int) list, month : int) =

    if null dates

    then []

    else

        if #2 (hd dates) = month

        then hd dates :: dates\_in\_month(tl dates, month)

        else dates\_in\_month(tl dates, month);

(\* 5 \*)

fun dates\_in\_months (dates : (int\*int\*int) list, months : int list) =

    if null months

    then []

    else dates\_in\_month(dates, hd months) @ dates\_in\_months(dates, tl months);

(\* 6 \*)

fun get\_nth (strings : string list, n : int) =

    if n = 1

    then hd strings

    else get\_nth (tl strings, n - 1);

(\* 7 \*)

fun date\_to\_string(date : int\*int\*int) =

    get\_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, numbers : int list) =

    let fun reaching\_sum(count: int, currSum: int, numbers : int list) =

        if (currSum + hd numbers >= sum)

        then count

        else reaching\_sum(count + 1, currSum + hd numbers, tl numbers)

    in

        reaching\_sum(0, 0, numbers)

    end;

(\* 9 \*)

fun what\_month(day: int) =

    number\_before\_reaching\_sum(day, [31, 28, 31, 30, 31, 31, 30, 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 (xs : (int\*int\*int) list) =

    if null xs

    then NONE

    else

    let fun oldest\_date\_nonempty (xs : (int\*int\*int) list) =

            if null (tl xs)

            then hd xs

            else

            let val tl\_ans = oldest\_date\_nonempty(tl xs)

            in

                if is\_older(hd xs, tl\_ans)

                then hd xs

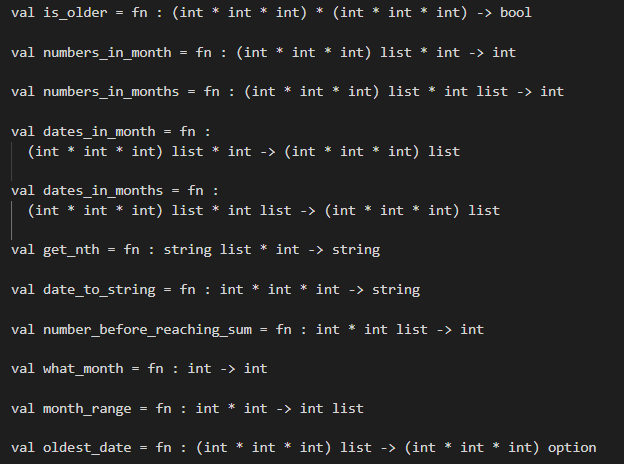
                else tl\_ans

            end

    in

        SOME (oldest\_date\_nonempty(xs))

    end;



Отже, всі функції компілюються правильно, приймають і повертають також вірний тип

**Тести:**

use "task.sml";

fun test(function\_name : string, true\_result, fact\_result) =

    if true\_result = fact\_result

    then (function\_name, "Ok")

    else (function\_name, "Failed");

(\* 1 \*)

test("is\_older", true, is\_older((2005, 1, 30), (2005, 1, 31)));

test("is\_older", false, is\_older((2010, 2, 2), (2005, 5, 31)));

test("is\_older", false, is\_older((2005, 1, 30), (2005, 1, 30)));

(\* 2 \*)

test("numbers\_in\_month", 3, numbers\_in\_month([(2005, 1, 30), (2005, 1, 30), (2005, 2, 30), (2005, 1, 30), (2005, 2, 30), (2005, 3, 30)], 1));

test("numbers\_in\_month", 0, numbers\_in\_month([(2005, 1, 30), (2005, 1, 30), (2005, 2, 30), (2005, 1, 30), (2005, 2, 30), (2005, 3, 30)], 5));

test("numbers\_in\_month", 1, numbers\_in\_month([(2005, 1, 30), (2005, 1, 30), (2005, 2, 30), (2005, 1, 30), (2005, 2, 30), (2005, 3, 30)], 3));

(\* 3 \*)

test("numbers\_in\_months", 4, numbers\_in\_months([(2005, 1, 30), (2005, 1, 30), (2005, 2, 30), (2005, 1, 30), (2005, 2, 30), (2005, 3, 30)], [1, 3]));

test("numbers\_in\_months", 2, numbers\_in\_months([(2005, 1, 30), (2005, 1, 30), (2005, 2, 30), (2005, 1, 30), (2005, 2, 30), (2005, 3, 30)], [2, 5]));

test("numbers\_in\_months", 0, numbers\_in\_months([(2005, 1, 30), (2005, 1, 30), (2005, 2, 30), (2005, 1, 30), (2005, 2, 30), (2005, 3, 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", "e4", get\_nth(["qwe1", "q2", "w3", "e4", "r5"], 4));

test("get\_nth", "qwe1", get\_nth(["qwe1", "q2", "w3", "e4", "r5"], 1));

test("get\_nth", "r5", get\_nth(["qwe1", "q2", "w3", "e4", "r5"], 5));

(\* 7 \*)

test("date\_to\_string", "February 28, 2022", date\_to\_string((2022, 2, 28)));

test("date\_to\_string", "April 2, 2010", date\_to\_string((2010, 4, 2)));

test("date\_to\_string", "December 12, 2012", date\_to\_string((2012, 12, 12)));

(\* 8 \*)

test("number\_before\_reaching\_sum", 3, number\_before\_reaching\_sum(7, [1, 2, 3, 4, 5]));

test("number\_before\_reaching\_sum", 2, number\_before\_reaching\_sum(6, [1, 2, 3, 4, 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", 12, what\_month(365));

(\* 10 \*)

test("month\_range", [], month\_range(30, 1));

test("month\_range", [1, 1, 2], month\_range(30, 32));

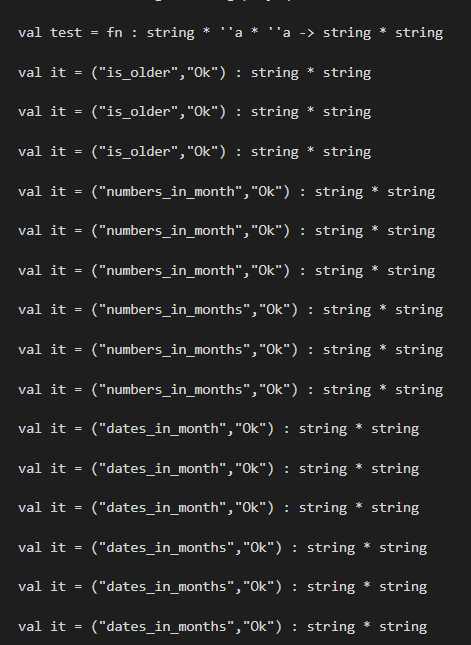
test("month\_range", [3, 3, 3], month\_range(70, 72));

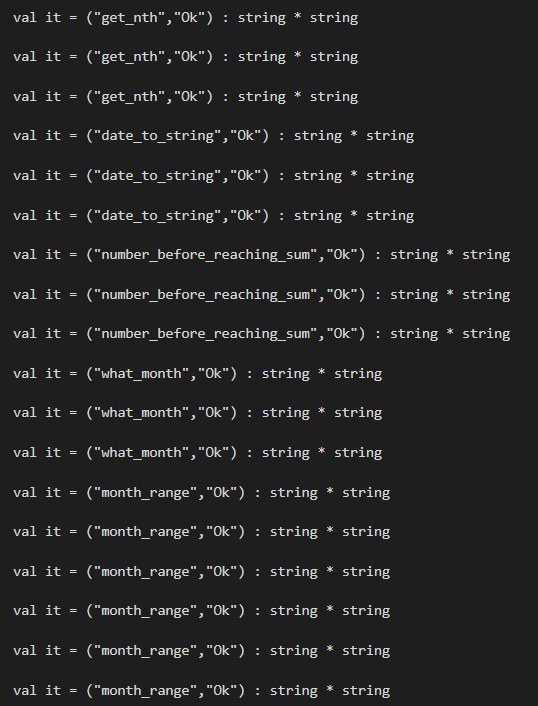
(\* 11 \*)

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)]));

****

****

Всі функції пройшли всі підготовлені тести