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;
```

```
val is_older = fn : (int * int * int) * (int * int * int) -> bool

val numbers_in_month = fn : (int * int * int) list * int -> int

val numbers_in_months = fn : (int * int * int) list * int list -> int

val dates_in_month = fn :
  (int * int * int) list * int -> (int * int * int) list

val dates_in_months = fn :
  (int * int * int) list * int list -> (int * int * int) list

val get_nth = fn : string list * int -> string

val date_to_string = fn : int * int * int -> string

val number_before_reaching_sum = fn : int * int list -> int

val what_month = fn : int -> int

val month_range = fn : int * int -> int list

val oldest_date = fn : (int * int * int) list -> (int * int * int) option
```

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)]));
```

```
val test = fn : string * ''a * ''a -> string * string

val it = ("is_older","Ok") : string * string

val it = ("is_older","Ok") : string * string

val it = ("is_older","Ok") : string * string

val it = ("numbers_in_month","Ok") : string * string

val it = ("numbers_in_month","Ok") : string * string

val it = ("numbers_in_month","Ok") : string * string

val it = ("numbers_in_months","Ok") : string * string

val it = ("numbers_in_months","Ok") : string * string

val it = ("numbers_in_months","Ok") : string * string

val it = ("dates_in_month","Ok") : string * string

val it = ("dates_in_month","Ok") : string * string

val it = ("dates_in_month","Ok") : string * string

val it = ("dates_in_months","Ok") : string * string

val it = ("dates_in_months","Ok") : string * string

val it = ("dates_in_months","Ok") : string * string
```

```
val it = ("get_nth","Ok") : string * string

val it = ("get_nth","Ok") : string * string

val it = ("get_nth","Ok") : string * string

val it = ("date_to_string","Ok") : string * string

val it = ("date_to_string","Ok") : string * string

val it = ("date_to_string","Ok") : string * string

val it = ("number_before_reaching_sum","Ok") : string * string

val it = ("number_before_reaching_sum","Ok") : string * string

val it = ("number_before_reaching_sum","Ok") : string * string

val it = ("what_month","Ok") : string * string

val it = ("what_month","Ok") : string * string

val it = ("what_month","Ok") : string * string

val it = ("month_range","Ok") : string * string

val it = ("month_range","Ok") : string * string

val it = ("month_range","Ok") : string * string

val it = ("month_range","Ok") : string * string

val it = ("month_range","Ok") : string * string

val it = ("month_range","Ok") : string * string
```

Всі функції пройшли всі підготовлені тести