Programming Assignment #2. OCaml Exercise

Prof. Jaeseung Choi

Dept. of Computer Science and Engineering

Sogang University

General Information

- **Check "HW #2" in *Assignment* tab of *Cyber Campus***
  - Skeleton code (`HW2.tgz`) is attached in the post
  - Deadline: **11/10** Fri. 23:59
  - Submission will be accepted in that post, too
  - Late submission deadline: **11/12** Sun. 23:59 **(-20% penalty)**
- **Please read the instructions in this slide carefully**
  - It contains important submission guidelines
    - If you do not follow the guidelines, you will get penalty

Skeleton Code

- **Copy HW2.tgz into CSPRO server and decompress it**
  - Don’t decompress-and-copy; copy-and-decompress
  - This course will use cspro**5**.sogang.ac.kr (**don’t miss the "5"**)
- **`P1/~P7/`: Directory for each problem**
  - 10 point for **P1**, and 15 point each for **P2 ~ P7**
- **`check.py`: Script for self-grading with test cases**
- **`config`: Used by the grading script (you don’t have to care)**

```
jason@ubuntu:~/OCaml-Exercise$ ls
check.py  config  P1  P2  P3  P4  P5  P6  P7
```

Before We Start

- **Once you adapt to the OCaml language, these exercise problems will be really easy**
  - Each of the problem can be solved with just few lines of code
- **And this programming assignment will take up 5% from the total score of the semester**
  - So take it as a ***gift***
- **Carefully read and understand the OCaml tutorial slide from our previous lecture: that will be enough**

Problem Directory Structure

- **Each directory will contain three files**
  1. Source file that contains the function you must fill in
     - For **`P1`**, it is **`sum_to_n.ml`**
  2. OCaml source file that contains the test code for this function
     - It starts with the prefix "**`test_`**"
  3. **`Makefile`** that compiles the two files into a binary executable
- **When you type "`make`" in this problem directory, it will generate an executable binary named `P1.bin`**
  - It also creates many by-products; type "**`make clean`**" to remove

```
jason@ubuntu:~/OCaml-Exercise/P1$ ls
Makefile  sum_to_n.ml  test_sum_to_n.ml
jason@ubuntu:~/OCaml-Exercise/P1$ make
ocamlc sum_to_n.ml test_sum_to_n.ml -o P1.bin
```

Specification

- **The requirement of each function that you have to write is simple and straightforward**
  - So the specification is directly written **in the comment**
  - Feel free to ask a question if you need a clarification of the spec

P1/sum_to_n.ml

```
(* Return the summation of integer from 0 to n.
 * Assume that n >= 0. *)
let rec sum_to_n n =
  1 (* TODO *)
```

Test Cases

- **The test cases in `test_*.ml` file can be also helpful**
  - In general, the test code will have a structure like below
  - "`check_testcase 5 15`": **5** is input and **15** is expected output

P1/test_sum_to_n.ml

```
open Sum_to_n

let check_testcase test_input answer =
  try
    if sum_to_n test_input = answer
    then Printf.printf "O"
    else Printf.printf "X"
  with _ -> Printf.printf "E"

let _ = check_testcase 5 15
```

Exception Handling in OCaml

- **In OCaml, you can catch exceptions using `try-with`**
  - `Division_by_zero`, `Not_found`, …
  - "_" for any exception (wildcard)
  - For each exception, define the value to return (cf. **`match-with`**)
- **You can also define or raise (throw) an exception**
  - I will not discuss it later if needed

```
let print_division x y =
  try Printf.printf "%d\n" (x / y) with
  Division_by_zero -> Printf.printf "Div-by-zero\n"

let find_from_map k m =
  try IntMap.find k m with
  | Not_found -> -1
```

Self-Grading

- **After filling in a function, you can compile and run the `P*.bin` file to see if the function works as you expected**
- **And if you think you have solved all the problems, run `check.py` as a final check**
  - `'O'`: Correct, `'X'`: Incorrect, `'E'`: exception, `'C'`: Compile error

    `'T'`: Timeout (maybe infinite recursion)

```
jason@ubuntu:~/OCaml-Exercise$ ./check.py
[*] Grading P1 ...
[*] Result: XXXX
[*] Grading P2 ...
[*] Result: XXX
[*] Grading P3 ...
[*] Result: XOX
```

Pre-defined Types

- **For some problems, pre-defined types are provided**
  - Do **NOT** change those types, just fill in the function below

```
(* DO NOT change the definition of this type *)
type exp =
    Num of int
  | Add of exp * exp
  | Sub of exp * exp
  | Mul of exp * exp
  | Div of exp * exp

(* Return the integer value represented by 'e'. *)
let rec eval e =
  match e with
  ...
```

Some (Syntax) Tips for P3

- **What kind of conditional expressions (syntaxes) exist in OCaml?**
  - Various comparisons, *And*, *Or*, *Not*, …

```
let b1 = 1 < 2
let b2 = 1 = 2   (* Caution: It's not "==" *)
let b3 = 1 <> 2  (* Caution: It's not "!=" *)

let b4 = b1 && b2
let b5 = b1 || b2
let b6 = not b1   (* Caution: Cannot use "!" *)

(* Caution: ^ does not mean XOR in OCaml *)
let b7 = b1 <> b2 (* For XOR, you can use <> *)
```

Submission Guideline

- **You should submit the following seven files (be careful not to submit compile by-product files like `*.cmo`)**
  - `sum_to_n.ml`
  - `exp_eval.ml`
  - `cond_eval.ml`
  - `sum_of_tree.ml`
  - `filter_list.ml`
  - `multiply_tree.ml`
  - `count_with_map.ml`
- **Submission format**
  - Upload these files directly to *Cyber Campus* (**do not zip them**)
  - **Do not change the file name** (e.g., adding any prefix or suffix)
  - If your submission format is wrong, you will get **-20% penalty**