COMP 3031 Assignment 1
SML programming
Fall 2017

Due: 5PM on 12 October 2017 (Thursday)

Instructions

- There are **five** questions in this assignment. Each question counts for two points. The total number of points is 10.
- Write your functions exactly the same as defined in the problem description (name, type, and functionality). In addition, you can write any helper functions as needed and call any built-in SML functions available in the lab machine.
- Put your entire solution in a single text file called "*ass1.ml*". In this file, put down your **name, ITSC account, and student ID** as a ***comment*** (surrounded by "(*" and "*)") on the first line.
- Submit your file through the Canvas before the deadline.
- Your submission will be tested under the SML interpreter on a lab machine by issuing the following command:
  - *use "ass1.ml";*

  Tips: to show the full list of nested data structures in sml, use the following statement:
  - *Control.Print.printDepth := 100;*
- **No** late submissions will be accepted under usual circumstances.

We define the following data types to be used throughout Assignment 1:

```
datatype course = C of (string * string) list;
datatype enroll = E of (int * string list) list;
```

A tuple in the datatype `course` contains two strings: the first one is the course id, and the second one is the id of the professor who teaches the course. A tuple in the datatype `enroll` consists of an integer (a student id), and a list of strings, which are the ids of the courses taken by the student.

Each course is taught by exactly one professor, and a professor may teach more than one course. All tuples in a `course` variable are distinct, and all tuples in an `enroll` variable are distinct. In each tuple of an `enroll` variable, all course ids in the string list are distinct.

Assume all data and user input are correct.

Question 1. Inserting courses

Write a function `insert_course` that returns a `course` with a given list of `(cid, pid)'s` inserted into the given course `C L`. If a `cid` already exists in the course ids in `L`, the function skips the tuple; otherwise, the tuple is appended to the end of `L`, in the order they appear in the given input tuple list.

```
val insert_course = fn : (string * string) list * course -> course
```

Examples:

```
- insert_course ([], C [("comp10", "p01")]);
val it = C [("comp10","p01")] : course

- insert_course ([("comp12", "p02")], C []);
val it = C [("comp12","p02")] : course

- insert_course ([("comp10", "p02")], C [("comp10", "p01")]);
val it = C [("comp10","p01")] : course

- insert_course ([("comp13", "p01"), ("comp12", "p02")], C [("comp10",
"p01")]);
val it = C [("comp10","p01"),("comp13","p01"),("comp12","p02")] :
course
```

Question 2. Inserting a student's enrollment

Write a function `insert_enroll` that returns an enroll variable with a given tuple `(sid, course_list)` inserted, where sid is the student id, and `course_list` is a list of distinct course ids. If the student id exists in the `enroll`, append the course ids in the given course list to the end of the course list of the student in the enroll, such that all course ids of the student are still distinct and the newly inserted course ids are in the same order as in the input list; otherwise, append the given tuple to the end of the enrollment list.

```
val insert_enroll = fn : (int * string list) * enroll -> enroll
```

Examples:

```
- insert_enroll ((1702, []), E [(1701, ["comp10", "comp11"])]);
val it = E [(1701,["comp10","comp11"]),(1702,[])] : enroll

- insert_enroll ((1701, ["comp10", "comp11"]), E []);
val it = E [(1701,["comp10","comp11"])] : enroll

- insert_enroll ((1701, ["comp11", "comp10"]), E [(1701, ["comp10",
"comp12"])]);
val it = E [(1701,["comp10","comp12","comp11"])] : enroll

- insert_enroll ((1702, ["comp10"]), E [(1701, ["comp10", "comp11"])]);
val it = E [(1701,["comp10","comp11"]),(1702,["comp10"])] : enroll
```

Question 3. Listing the students enrolled in a course

Write a function `query_students` that returns a list of ids of all the students enrolled in a course with the given course id. The order of student ids in the returned list is the same as the order they appear in the `enroll`.

```
val query_students = fn : string * enroll -> int list
```

Examples:

```
- query_students ("comp10", E []);
val it = [] : int list

- query_students ("comp10", E [(1701, ["comp10"])]);
val it = [1701] : int list

- query_students ("comp11", E [(1701, ["comp10"])]);
val it = [] : int list

- query_students ("comp10", E [(1701, ["comp10", "comp11"]), (1702,
["comp13", "comp10"]), (1703, [])]);
val it = [1701,1702] : int list
```

Question 4. Counting the number of distinct students enrolled in a professor’s courses

Write a function `count_distinct_students` that returns the number of distinct students enrolled in all the courses taught by the professor `pid`.

```
val count_distinct_students = fn : string * course * enroll -> int
```

Examples:

```
- count_distinct_students ("p01", C [], E []);
val it = 0 : int

- count_distinct_students ("p01", C [("comp10", "p01")], E []);
val it = 0 : int

- count_distinct_students ("p01", C [("comp10", "p01")], E [(1701,
["comp10"]), (1702, ["comp13"])]);
val it = 1 : int

- count_distinct_students ("p01", C [("comp10", "p01"), ("comp12",
"p02"), ("comp13", "p01")], E [(1701, ["comp10", "comp11"]), (1702,
["comp13"])]);
val it = 2 : int
```

Question 5. Deleting a course

Write a function `delete_course_enroll` that deletes a course of a given course id and its corresponding enrollments. This function returns a tuple consisting of an updated `course` and an updated `enroll`. The order of the tuples in the output remains the same as the order before the deletion.

```
val delete_course_enroll = fn : string * course * enroll -> course *
enroll
```

Examples:

```
- delete_course_enroll ("comp10", C [], E[]);
val it = (C [],E []) : course * enroll

- delete_course_enroll ("comp10", C [("comp10", "p01")], E []);
val it = (C [],E []) : course * enroll

- delete_course_enroll ("comp10", C [("comp10", "p01")], E [(1701,
["comp10"])]);
val it = (C [],E [(1701,[])]) : course * enroll

- delete_course_enroll ("comp10", C [("comp10", "p01"), ("comp12",
"p02")], E [(1701, ["comp10", "comp11"]), (1702, ["comp13", "comp10"]),
(1703, [])]);
val it =
  (C[("comp12","p02")],E [(1701,["comp11"]),(1702,["comp13"]),(1703,[])])
  : course * enroll
```