Homework: Lambda Calculus & Racket

Due-date: Feb 09 at 11:59pm; Submit online on Blackboard LS
(**Q1 solution in pdf format and Q2 solution as one Racket file**)

Homework must be individual's original work. Collaborations and discussions of any form with any students or other faculty members are not allowed. If you have any questions and/or concerns, post them on Piazza and/or ask our TA or me.

Learning Outcomes

- Application of knowledge of computing and mathematics
- Ability to understand the implications of mathematical formalisms in computer science
- Knowledge and application of Functional Programming
- Ability to follow requirement specification

Questions

1. Given

   0: $\lambda f.\lambda x.x$

   $succ$: $\lambda n.\lambda f.\lambda x.(f\ ((n\ f)\ x))$

   n: if n is a natural number then its semantics is the result of n applications of $succ$ on 0.

   $true$: $\lambda x.\lambda y.x$

   $false$: $\lambda x.\lambda y.y$

   $second$: $\lambda x.\lambda y.\lambda z.y$

   g: $\lambda n.((n\ second)\ false)$

   What is the result of

   (a) $(g\ n)$ when n is 0.

   (b) $(g\ n)$ when n results from some application $succ$ on 0.

   (c) What mathematical/logical operation is computed by g?

   (5)

2. Consider the following λ-expression:

   $$Y:\ \lambda t.(\lambda x.(t\ (x\ x))\ \ \lambda x.(t\ (x\ x)))$$

   Prove/disprove that $(Y\ t)$ after application of several β-reductions result in $(t\ (Y\ t))$. (5)

3. As part of this question, we will use certain naming convention explained below. Consider the function (we have discussed this function in class):

```
(define myapply
  (lambda (f)
    (lambda (lst)
      (if (null? lst)
          lst
          (cons (f (car lst)) ((myapply f) (cdr lst)))))))
```

The function is designed to "apply" some function f on each element of a given list. I.e., the assumption is that the `lst` is a list.

We will say that the function `myapply` contains two λ-parameters: a function f and a list such that the function f can be applied to elements in the list.

Consider a database of students and course records presented in the form of Racket-list structure. The syntax of the records is as follows

```
student-table    -> (list-of-students)
list-of-students -> student-record | student-record list-of-students
student-record   -> (student-id  student-name course-plan)
student-id       -> number
student-name     -> symbol
course-plan      -> (list-of-courses)
list-of-courses  -> course-number | course-number list-of-courses
course-number    -> number

grade-table      -> (list-of-grades)
list-of-grades   -> grade-record | grade-record list-of-grades
grade-record     -> (course-number student-id grade)
grade            -> A | B | C | D | F
```

An instantiation of database following the above grammar is presented in the file DB.rkt[1]. It contains the following:

```
#lang racket
(provide (all-defined-out))
(require racket/trace)

(define student-table
  '( (0 Asterix (342 230 412 227))
     (1 Obelix  (342 228 227 230))))

(define grade-table
  '( (342 0 A)
     (227 0 B)
     (412 0 A)
     (227 1 C)
```

[1]For grading, we will use other instantiations of the database.

```
(228 1 A)
(342 1 B)
))
```

You are required to write several functions in Racket that corresponds to querying the database as per query-specification. You will submit one Racket file for this question. It must be named `<yournetid>.rkt`. It must contain the following directives at the top of the file

```
#lang racket
(provide (all-defined-out)) ;; for us to test
(require racket/trace)      ;; in case you want to use tracing
(require "DB.rkt")          ;; to access the database definitions in DB.rkt
```

You can/will use the operations such as null?, list?, cons, list, car, cdr, equal? and control constructs such as if-then-else and recursion. You must not use let, let, set!, get!, begin-end; or any other construct that allows you to declare variables, sequence operations–using any of these may result in 0 points in this question. If you are in doubt whether or not you can use some construct or operation, please ask us.

(a) Write a function `applyonstds`, which contains two λ-parameters: `student-table` and f, any function that can be applied to the student-record. The function `applyonstds` results in the application of f on each student-record.

(b) Write a function `numberofcourses`, which contains one λ-parameter: `student-record`. It results in a record where course-plan in the student-record is replaced by the number of courses in the course-plan.

The following shows the invocation pattern and result of the invocation

```
> ((applyonstds student-table) numberofcourses)
'((0 Asterix 4) (1 Obelix 4))
```

This is similar to SQL Query

```
select student-id, student-name, len(course-plan) as numberofcourses
from student-table
```

(c) Write a function `studentgpa`, which contains two λ-parameters: `grade-table` and `student-record`. It returns the student record, where the course-plan in the record is replaced by the GPA secured by the student. Note that, student may not have taken all courses in the course plan; therefore, the grade-table may not have entries for all courses in the course plan of a student.

```
> ((applyonstds student-table) (studentgpa grade-table))
'((0 Asterix 3.67) (1 Obelix 3.0))
```

(d) Write a function `gradebook`, which contains three λ-parameters: course id, grade-table and a function f that can be applied to student id. It returns the list of records for the students who have taken the course (specified by course id) in the following list-form: the result of application of function f on student-id followed by the grade of the student in the course.

For instance, if there is a function `findstudentwithgpa` with three λ-parameters: student-table, grade-table and student id, which returns a list containing the student id, the student name, student GPA of the student (specified by the student id), then

```
> (((gradebook 342) grade-table) ((findstudentwithgpa student-table)
                                                        grade-table))
'((0 Asterix 3.67 A) (1 Obelix 3.0 B))
```

(5 + 5 + 5 + 10)

You may have to write helper functions (such as converting list of grades to GPA). For every function you write, please write in comments the type of input, the result-type and present example of expected results. For testing, you should try to test each function separately before testing their composition. To get you started, here is an example query implementation: objective is to find the list of students (id and names) who have a specified course in their plan.

```
;; s-table: student-table
;; course: course id
;; review the usage
;; example: ((planincludes student-table) 342): '((0 Asterix) (1 Obelix))
;; example: ((planincludes student-table) 113): '()
;; example: ((planincludes student-table) 412): '((0 Asterix))
(define planincludes
  (lambda (s-table)
    (lambda (course)
      (if (null? s-table)
          s-table        ;; nothing to do

          (if (presentin (cadr (cdr (car s-table))) course)
              ;; presentin: true
              ;; add the student information using "cons"
              (cons (list (car (car s-table)) (cadr (car s-table)))
                    ((planincludes (cdr s-table)) course))

              ;; else move on without adding the student information
              ((planincludes (cdr s-table)) course))))))

;; Not using lambda parameters as I don't need to
;; lst: list of courses
;; course: course id
;; example: ((presentin '(1 2 3)) 2): true
;; example: ((presentin '(1 2 3)) 0): false
(define (presentin lst course)
  (if (null? lst)
      false
      (if (equal? (car lst) course)
          true
          (presentin (cdr lst) course))))
```