Problem Set 6
Comp 301
Fall 2022
Week 7: 21.11.2022 - 25.11.2022

Instructions:

- Submit your answers to the Blackboard PS6 assignment until November 26th Saturday, at 23.59.
- Please use the code boilerplate for `PROC` language, which includes several tests for you to see if your code is correct.
- Save your code and zip it as `ID_username.zip` with your ID and username (Example: `1234567_yakarken18.zip`), and submit this ZIP file.
- Read the questions carefully. Good luck!

Part A:. Calculation of a program in PROC language using the specification is shown below. However, it is not complete. Fill in the blanks where environment is suppose to be. What will this program return? Explain briefly how you found your answer.

```
(value-of
   <<let a = 10 in
      let f = proc (func) (func a) in
         let a = 5 in
            let g = proc (x) -(x, a) in
               (f g)>>
ρ)

= (value-of
   <<let f = proc (func) (func a) in
         let a = 5 in
            let g = proc (x) -(x, a) in
               (f g)>>
_______________)

= (value-of
   <<let a = 5 in
            let g = proc (x) -(x, a) in
               (f g)>>
_______________
_______________)

= (value-of
   <<let g = proc (x) -(x, a) in
               (f g)>>
_______________
_______________
_______________)

= (value-of <<(f g)>>
_______________
_______________
_______________
```

_____)

Part 2 (Coding):

Problem 1: Add my-cond to the language. Unlike cond of the LET language, the evaluation steps are a little bit different in my-cond. In cond of the LET language, starting from the first condition it evaluates expressions in order and the true condition will be evaluated as a result. However, now you need to implement my-cond a little bit different:

- starting from the first condition it evaluates expressions in order.
- this time you return the last true evaluated condition as a result, instead of returning the first condition evaluated as true.

To better understand how my-cond should be working, there is an example provided. For example,

```
(let ((x 2) (y 6) (z 7))
     (cond ((> x 4) 12)
     ((< x y) 13)
     ((< y z) 15)
     (else 11)))

;; returns 15 instead of 13!!
```

So, second and third conditions are evaluated as true but it should return the last condition evaluated as true, it returns

```
(< y z 15) ;; returns 15.
```

The syntax for my-cond is given below. Implement the procedure accordingly.

```
Expression ::= my-cond Expression then Expression,
                {Expression then Expression,}*
                else Expression
```

my-cond (cond1 exp1 conds exps else-exp)

Problem 2: Modify the proc procedure and add necessary test cases.

Hint: For part A you need to modify tests.rkt. For Part B, you have to change lang.rkt and interp.rkt.

Part A. [1] In PROC, procedures have only one argument, but one can get the effect of multiple argument procedures by using procedures that return other procedures. For example, one might write code like

```
let f = proc (x) proc (y) ...
in ((f 3) 4)
```

This trick is called Currying, and the procedure is said to be Curried. Write a Curried procedure that takes two arguments and returns their sum to specified places in test.rkt. You have to write x + y in terms of subtraction because of constrains of our language.

[1] EOPL p.80 Exercise 3.20

Part B. [2] Extend the `PROC` to include procedures with multiple arguments and calls with multiple operands, as suggested by the grammar:

```
Expression ::= proc({Identifier}(,)) Expression
           ::= (Expression{Expression})
```

Here is an example usage which evaluates to 1:

```
let f = proc(x, y) -(x,y)
in (f 5 4)
```

After you finish implementing the described operations, uncomment and run the tests in tests.rkt.

[2] EOPL p.80 Exercise 3.21