**CSSE 304 Final Exam Part 3 (computer part)** Wednesday evening, Feb 20, 2019 **Name\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

You may not use code written by other CSSE 304 students (from this term or any previous term), or code that you find on the internet. **You are not allowed to use mutation for either problem.**

You may receive some partial credit based on your code and comments. If your code passes the test cases but does not implement the required features according to the specification, you may receive less than full credit.

**Cautions!** Do not get so caught up in getting all of the points for one problem that you do not have time to work on the other one.

Do not write code that is specific to my test-cases but does not solve the general problem.

**There are two different assignments on the PLC server; one for each problem**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Problem** | **Max score** | **Your score** | **Instructor/grader comments** |
| transitive-closure | 1 | 13 |  |  |
| pascal-triangle-cps | 2a | 12 |  |  |
| pt-imp (extra credit) | 2b | 10 |  |  |
|  | **Total** | **25** |  |  |

1. Recall our representation of a relation by a set of two-lists (a two-list is a list of two items). Example: ((a b) (a c) (b c)). But ((a b) (b c) (ab)) does not represent a relation because of the duplication.  
   A relation is *transitive* iff, for any symbols a, b, and c, whenever (a b) and (b c) are in the relation, so is (a c) .  
   Some examples of transitive relations: () ((a a)) ((a a) (b a) (a c) (b c))  
   The relation ((a a) (b a) (a c)) is not transitive because it does not contain (b c).   
     
   The *transitive closure* of a relation R is the smallest transitive relation that contains all of the pairs in R. Define a Scheme procedure transitive-closure that takes a relation as its argument and returns the transitive closure of that relation. The order of 2-lists in the list your procedure returns does not have to match the order shown in answers for my test-cases.

**Examples:**

> **(transitive-closure '())**

()

> **(transitive-closure '((a a) (a b)))** ; an answer of ((a b) (a a)) is also correct for this case.

((a a) (a b))

> **(transitive-closure '((a a) (a b) (b c)))**

((a a) (b c) (a b) (a c))

> **(transitive-closure '((a b) (b c) (d a)))**

((a b) (b c) (d a) (a c) (d c) (d b))

> **(transitive-closure '((a b) (b c) (d a) (c a)))**

((a c) (b a) (a b) (d a) (d c) (c a) (c c) (b c) (b b) (a a) (c b) (d b))

> **(transitive-closure '((a b) (b c) (a c)))**

((a b) (b c) (a c))

> (**transitive-closure '((a b) (b c) (d a) (c a) (e d)))**

((e d) (a c) (b a) (a b) (d a) (d c) (c a) (c c) (e c) (e b) (b c) (b b) (a a) (e a) (c b) (d b))

> **(transitive-closure '((a b) (c b)))**

((a b) (c b))

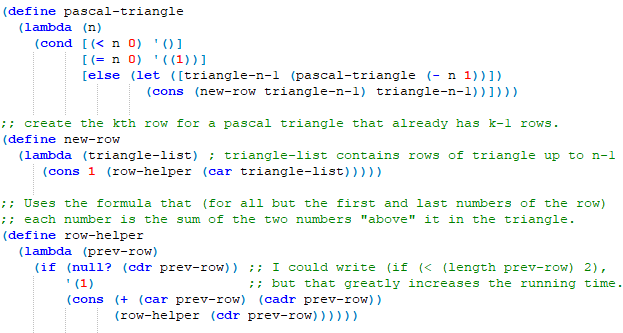
> **(transitive-closure '((a a) (a b) (b a)))**

((a b) (a a) (b b) (b a))

> **(transitive-closure '((a b) (b a)))**

((a b) (b a) (a a) (b b))

1. My solution to **pascal-triangle** (same solution that I presented in class) appears here and at the top of the test-case file.

If you wish, you can re-read the description of that Assignment 4 homework problem. 

Convert my code to CPS. You must rewrite my procedures in CPS, rather than substituting your own solution. You are allowed to add new helper procedures (including apply-k, of course) as appropriate. All **applications** of substantial procedures (including pascal-triangle-cps, new-row-cps, row-helper-cps, and apply-k) must be in tail position.   
**Reminder:** Continuation constructors are not substantial.

You must provide two initial continuations, init-k and car-k. Applying the car-k continuation returns the car of the argument. Applying the init-k continuation returns the argument. The examples show how these will be used in the test cases.  
  
For full credit, you must use our data structures representation of continuations (defined using define-datatype). You can receive half-credit for CPS code that passes all of the test cases and represents continuations by Scheme procedures as in A15.

**Examples:**

> **(pascal-triangle-cps 4 (init-k))**

((1 4 6 4 1) (1 3 3 1) (1 2 1) (1 1) (1))

> **(pascal-triangle-cps 12 (car-k))**

(1 12 66 220 495 792 924 792 495 220 66 12 1)

**Extra-credit problem** 2b. Convert a copy of your pascal-triangle-cps code to imperative form. So that the PLC server can distinguish this version from the previous version, call your new procedure pt-imp. All substantial procedures that pt-imp calls must also be written in imperative form.   
Suggestion: give the other substantial procedures names that end in -imp.   
**You must define global variables** n and k (along with any other variables that you need to make the imperative-form version work).

**Examples:  
> (begin**

**(set! k (init-k))**

**(set! n 5)**

**(pt-imp))**

((1 5 10 10 5 1) (1 4 6 4 1) (1 3 3 1) (1 2 1) (1 1) (1))

**> (begin**

**(set! k (car-k))**

**(set! n 9)**

**(pt-imp))**

(1 9 36 84 126 126 84 36 9 1)