TODO Chapter 5 [15/26]

1. CPS Interpreter [10/16]
   (a) [ ] CPS Lambda Style
   (b) [X] CPS Data Rep
   (c) [X] let2 (generalized in 5.7)
   (d) [X] let3 (generalized in 5.7)
   (e) [X] list extendsion
   (f) [X] list keyword (sophisticated)
   (g) [X] multiple declaration let
   (h) [X] multiple declaration lambda
   (i) [X] implicit reference langauge (same with let expression)
   (j) [X] implicit reference langauge (change binding to other storage)
   (k) [X] begin expression
   (l) [ ] output information (IO Monad makes printing problematic) Pending
   (m) [ ] fact `fact-iter`
   (n) [ ] profile `fact-iter` and `fact` (IO Monad)
   (o) [ ] list continuation representation (`StateT` / `ReaderT`)
   (p) [ ] statement extension (to be implemented in monadic interp)
2. Trampolined Interpreter (tail recursion) [3/6]
   (a) [@17] [X] wrap `Bounce` type around `applyProcedureK`
   (b) [X] `data` representation of `Bounce`
   (c) [X] wrap it around `applyCont` will have not type change
   (d) [ ] optimize ending `applyCont KEmpty refVal`
   (e) [ ] implement in procedural language (replacing trampoline as loop) (to be implemented in code generation)
   (f) [ ] to be verified later
3. Imperative Interpreter (State Monad, Skipped) [0/0]
4. Exception [2/4]
   (a) [@35] [ ] Direct Access to apply-handler (Omitted for a while)
      - Add one more try layer
      - Memorize try-cont
      - `StateT` Monad
   (b) [ ] use two continuation to deal with exceptions
   (c) [X] call with wrong number of arguments
      - Partial Application was implemented
   (d) [X] division and `divbyzero`

TODO Chapter 7 (Types) [0/1]

1. [ ] What types do value of following expressions have?
   (a) `\x -> x - 3 : int -> int`
   (b) `\f -> \x -> (f x) - 1 : ('a -> int) -> 'a -> int`

(c) `\x -> x` : `'a -> 'a`

(d) `\x -> \y -> x y` : `('a -> 'b) -> 'a -> 'b`

(e) `\x -> (x 3)` : `(int -> 'a) -> 'a`

(f) `\x -> (x x)` : infinite type `t -> t`

(g) `\x -> if x then 88 else 99` : `bool -> int`

(h) `\x -> \y -> if x then y else 99` : `bool -> int -> int`

(i) `(\p -> if p then 99 else 99) 33` : type error

(j) `(\p -> if p then 99 else 99) (\z -> z)` : type error

(k) `\f -> \g -> \p -> \x -> if (p (f x)) then (g 1) else (f x) - 1 : ('a -> int) -> (int -> int) -> (int -> bool) -> 'a`

(l) `\x -> \p -> \f if (p x) then x - 1 else (f p)`

:

```
Int -> (Bool -> Int) -> ((Bool -> Int) -> Int)
```

2.