CSE 114A Midterm 1, Winter 2024

NAME : _____

CruzID: _____@ucsc.edu

- DO NOT TURN THIS PAGE OVER BEFORE WE TELL YOU TO
- You have **90 minutes** to complete this exam.
- Where limits are given, **write no more** than the amount specified.
- You may refer to a **double-sided cheat sheet**, but no electronic materials.
- Avoid seeing anyone else's work or allowing yours to be seen.
- Do not communicate with anyone but an exam proctor.
- If you are unsure of how to interpret a problem description, state your interpretation clearly and concisely. Reasonable interpretations will be taken into acount by the graders.
- **Good luck!**

Q1: Scope[10 pts]

For each bound occurrence of a variable in the following lambda terms, draw an arrow pointing to its binder. For each free occurrence, draw a circle around the variable.

```
(\x -> \y -> \z -> x (\y -> z (\z -> w z)) y)
```

```
(\x y z -> (\a b c -> x y c)) (\x y z -> a y c)
```

Q2: Reductions [10 pts]

For each λ-term below, check the box next to **each** valid reduction of that term. It is possible that none, some, or all of the listed reductions are valid. Reminder:

- `=a>` stands for an α-step (α-renaming)
- `=b>` stands for a β-step (β-reduction)
- `=~>` stands for a sequence of *zero or more* steps, where each step is either an α-step or a β-step, and the right-hand side is in *normal form*

2.1 [5 pts]

```
(\x -> x x) (\x -> x x)
```

(A) `=b> \x y -> y x` [ ]

(B) `=b> (\x -> x x) (\x -> x x)` [ ]

(C) `=b> \x y -> (\x -> x (x y))` [ ]

(D) `=a> (\x -> x x) (\y -> y y)` [ ]

(E) `=a> \x y -> (\z y -> y z) (x y)` [ ]

2.2 [5 pts]

```
(\x -> x) (\y -> apple y) (\z -> z)
```

(A) `=b> (\x -> x) (apple (\z -> z))` [ ]

(B) `=b> (\y -> apple y) (\z -> z)` [ ]

(C) `=a> (\z -> z) (\y -> apple y) (\z -> z)` [ ]

(D) `=a> (\x -> x) (\y -> orange y) (\z -> z)` [ ]

(E) `=~> apple (\z -> z)` [ ]

Q3: Factorial [10 pts]

In this task you will implement the *factorial function* in lambda calculus. Your implementation of `FACT` should satisfy the following test cases:

```
eval fact0 :    eval fact1 :    eval fact2 :    eval fact3 :
  FACT ZERO       FACT ONE        FACT TWO        FACT THREE
  =~> ONE         =~> ONE         =~> TWO         =~> SIX
```

You can use any function defined in the "Lambda Calculus Cheat Sheet" at the end of this exam, including the fixpoint combinator `FIX`. You should define a helper function `STEP`.

```
let STEP = ____________________________________________________

           ____________________________________________________

let FACT = ____________________________________________________

           ____________________________________________________
```

Q4: Haskell Values and Patterns [10 pts]

Q4.1 [5 pts]

In Haskell, what is the definition of a value?

Q4.2 [5 pts]

In Haskell, what is the definition of a pattern?

Q5: Haskell Types [10 pts]

Fill in the blanks to show the Haskell type of each of the following expressions.

Q5.example

```
True :: Bool
```

Q5.1

```
(True,"abc") :: ____________________________________
```

Q5.2

```
["def","abc"] :: ____________________________________
```

Q5.3

```
(True : False : []) :: ____________________________________
```

Q5.4

```
[] :: ____________________________________
```

Q5.5

```
(\x -> if x then "hello" else "goodbye") :: _________________________________
```

Lambda Calculus Cheat Sheet

Here is a list of definitions you may find useful for Q3

```
-- Booleans --------------------------------

let TRUE  = \x y -> x
let FALSE = \x y -> y
let ITE   = \b x y -> b x y

-- Pairs -----------------------------------

let PAIR  = \x y b -> b x y
let FST   = \p     -> p TRUE
let SND   = \p     -> p FALSE

-- Numbers ---------------------------------

let ZERO  = \f x -> x
let ONE   = \f x -> f x
let TWO   = \f x -> f (f x)
let THREE = \f x -> f (f (f x))
let FOUR  = \f x -> f (f (f (f x)))
let FIVE  = \f x -> f (f (f (f (f x))))
let SIX   = \f x -> f (f (f (f (f (f x)))))

-- Arithmetic ------------------------------

let INC   = \n f x -> f (n f x)
let ADD   = \n m -> n INC m
let MUL   = \n m -> n (ADD m) ZERO
let ISZ   = \n -> n (\z -> FALSE) TRUE
let SKIP1 = \f p -> PAIR TRUE (ITE (FST p) (f (SND p)) (SND p))
let DEC   = \n   -> SND (n (SKIP1 INC) (PAIR FALSE ZERO))

-- Recursion -------------------------------

let FIX   = \stp -> (\x -> stp (x x)) (\x -> stp (x x))
```