CSCI-GA - Problem Set 5 - Written Part

Task 1.1

1.
   - When type-checking `let` we get the type of the symbol immediately along with the symbol name, as the let expression introduces a local variable bound to a specific value.
   - For e.g.
     - ```
       (let ([a 3]) (+ a 7))
       ```
     - here the type of a is immediately available (type "number" / "numT (as per convention in lecture 12)")
   - Hence, when we add the let-binding into the type-environment, we have both the symbol-name and symbol-value available with us.
2. When type-checking `lambda` expressions, we do not immediately get the type of the symbol / variable being used in the body of the lambda expression. Hence, the only way to determine the type of the symbol used in the lambda expression is via complex "type-inference" (wherein we determine the type of the symbol, by checking the all the expressions in the body of the lambda expression and figuring out which type of symbol would fit those conditions) which has considerable algorithmic complexity.

please turn over
       ```

Task 2.1

1. f1

```
(define f1
    (let
        ([b (box empty)])
        (lambda (x) (set-box! b (cons x (unbox b))))))
```

   - type: `('_a -> void)`
   - `'_a` '_a in PLAI-typed represents a type variable that can be instantiated with a specific type. Thus f1 accepts values of some fixed but unspecified type that shall stay the same across multiple function calls due to the box being defined outside the body of the lambda expression.

2. f2

```
(define f2
    (lambda (x)
        (let
            ([b (box empty)])
            (set-box! b (cons x (unbox b))))))
```

   - type: `('a -> void)`
   - `'a` indicates a type variable of unspecified type. Thus, f2 accepts values of any type. This is because the box is defined within the body of the lambda expression, and thus is scoped to the lambda expression.

3. Why does `(begin (f1 #t) (f1 1))` cause a type error?

   - as f1 accepts values of some fixed but unspecified type, during the second call the type of input f1 expects has already been defined as boolean type. Hence there is a type mismatch on passing argument `1` to f1 causing a type error.

4. Why does `(begin (f2 #t) (f2 1))` not give any type error?

   - as f2 accepts values of any type (as the box is defined within the body of the lambda expression used within f2), we can pass different types of arguments to f2 with each call.