Question 1: Theoretical Questions

Q1.1 Why are special forms required in programming languages? Why can't we simply define them as primitive operators? Give an example.

Special forms are required in case the semantics of the evaluation does not follow the default 'procedure application' semantics, *i.e.,* evaluation of the operator and the operands + application. The 'if' special form, for example, does not required pre-evaluation of the 'then' and the 'else' sub-expressions. In case 'if' was a primitive operator rather than a special form, the evaluation of the following expression would cause an error: (if (> 3 4) 5 (\ 6 0))

Q1.2 Write a program in L1, containing more than one expression, where the evaluation of the program's expression **can be done in parallel** (e.g., the interpreter can run a thread for each expression evaluation).
Write a program in L1, containing more than one expression, where the evaluation of the program's expression **cannot be done in parallel**.

Can be done in parallel:
(+ 3 4)
(* 7 8)

Cannot be done in parallel:
(define x 5)
(* 7 x)

Q1.3 Let us define the L0 language as L1 excluding the special form 'define'. Is there a program in L1 which cannot be transformed to an equivalent program in L0? Explain or give a contradictory example [4 points]

No. Since there are no recursive forms in L1 (e.g., user procedures), any var reference can be replaced by its defined value expression.

Q1.4 Let us define the L20 language as L2 excluding the special form 'define'. Is there a program in L2 which cannot be transformed to an equivalent program in L20? Explain or give a contradictory example.

Yes. L2 supports user procedures which may contain recursion calls (a variable which is defined as a lambda may appear in the body of the lambda). In this case, var references cannot be replaced by their defined value.

Q1.5 For the following high-order functions in L3, which get a function and a list, indicate (and explain) whether the order of the procedure application on the list items should be sequential or can be applied in parallel:

- map
- reduce
- filter
- all (returns #t is the application of the given boolean function on each of the given list items returns #t)
- compose (compose a given procedure with a given list of procedures)

L3 is pure functional, i.e., there are no side-effects (beside the special *define* form, which is not CExp)

- map
  Can be applied in parallel. The evaluation of the function application on one item has no effect on the evaluation application of the function on another item.

- Reduce
  The reducer application is based on the reduce of the previous items. Can be applied in parallel just in case the reducer procedure is commutative and associative (e.g., +)

- Filter
  Can be applied in parallel. The evaluation of the predicate application on one item has no effect on the evaluation application of the function on another item.

- all (returns #t is the application of the given boolean function on each of the given list items returns #t)
  Can be applied in parallel. The evaluation of the predicate application on one item has no effect on the evaluation application of the function on another item. In addition, *and* operator is commutative and associative.

- compose (compose a given procedure with a given list of procedures)
  Can be applied in parallel just in case the procedure composition is associative.

Q1.6 Regarding L31 language, as defined in **Q3b** (below): what is the value of the following program? Explain.

```
(define b 1)
(define c 2)
(define pair
    (class (a b)
         ((first (lambda () a))
          (second (lambda () b))
          (f (lambda () (+ a b c)))
    )
)

(define p34 (Pair 3 4))

(
   (lambda (c) (p34 'f))
   5
)
```

pair is actually a closure which contains the values of c (2), where b is bounded, un contrast, to the b parameter.
p34 is actually a closure which contains the values of a (3) and b (4), in addition to the value of c (2).
The application of `(lambda (c) (p34 'f))` with the substitution of c : 5, has no effect on the value of c in the closure of p34, which was bounded to 1 during its creation.
→ `a + b + c = 9`