Please write your name at the top of this cover page and nowhere else.

A crib sheet is permitted according to our agreed-upon rules (both sides of one piece of paper, etc.). Otherwise, no notes, books, phones, calculators, computers, etc. are allowed.

If you cannot understand what a question is asking, then ask for clarification. If you cannot obtain clarification, then include your interpretation of the problem in your solution. Never interpret a problem in a way that renders it trivial.

You are required to explain your reasoning or otherwise show your work, so that I can understand how you arrived at your answer. Incorrect answers with solid work often earn partial credit. Correct answers without explanatory work rarely earn full credit.

Any question that refers to OOC is referring to its final version (version h).

You have 150 minutes (two and a half hours).

Good luck. :)

This Scheme code is shown in two columns, just to save page space.

```
(define helper
  (lambda (lyst n)
    (if (null? lyst)
        n
        (helper (cdr lyst) (+ n 1)))))
```

```
(define list-length
  (lambda (lyst)
    (helper lyst 0)))
```

A.A. Is the `helper` function tail-recursive? Explain, of course.

A.B. Draw all of the frames involved in Guile's evaluation of `(list-length '(2 3 5))`.

B.A. I'm designing a resizable `Array` class in OOC. The class definition begins with the code chunk below. The `data` member is essentially an array of `void *`s, which gets allocated and reallocated as necessary. What C code does the precompiler emit, based on this code chunk?

```
^^superclass Object
^^member int length;
^^member void **data;
```

B.B. Later in the class definition I declare the following method to set the `i`th object in the array. What lines of code should go in the blank space? (Hint: I have four lines in there, and none of them are error checks.)

```
// Sets the object at index i of the array to obj. Use as if the prototype were
//     void *set(void *self, Integer *i, void *obj);
^^method void *set(void *self, va_list *args) {
    Array *array = (Array *)self;
    Integer *i = va_arg(*args, Integer *);
    void *obj = va_arg(*args, void *);
    assert(array != NULL && i != NULL && obj != NULL);
    assert(i->value >= 0 && i->value < array->length);



    return self;
}
```

C.A. In OOC, what does the `new()` function do, before it calls the `initialize()` method?

C.B. Why is that work done in `new()` rather than in `initialize()`?

D. Give an example of duck typing from our OOC code.

The three-part problem below is about our Scheme interpreter project's C code. Your answers should be detailed enough, to demonstrate the differences (if any) among the three parts.

E.A. How does the evaluator handle expressions of the form `(func ...)`, where `func` is previously `define`d by the user to be the value of a `lambda` expression?

E.B. How does the evaluator handle expressions of the form `(+ ...)`?

E.C. How does the evaluator handle expressions of the form `(if ...)`?

F. In OOC, suppose that we end up with a deep class hierarchy, where `ClassZ` subclasses `ClassY`, which subclasses `ClassX`, ..., which subclasses `ClassA`, which subclasses `Object`. And none of these classes override `Object`'s `retain` method. And an instance of `ClassZ` is `retain`ed. What is bad about this situation?

G. As we've discussed, CPython mainly uses reference counting to manage its object graph. (The reference counting is automated, so that the Python user-programmer doesn't have to do it.) Additonally, CPython periodically runs a generational garbage collector. Why do you think CPython uses this mix of strategies (as opposed to, say, just one of them)?