You have 70 minutes.

No notes, books, 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.

Unless a problem tells you that no explanation is needed, 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.

Good luck. :)

A.A. In Scheme, is `let` a keyword or a function? Explain, of course.

A.B. Rewrite this `let` expression to use `lambda` rather than `let`. No explanation is needed.

```
(let ((var1 val1) (var2 val2)) body)
```

Recall from homework that a lazy list is either `()` (if it's empty) or a dotted pair `(a . f)`, where `a` is the first element of the list and `f` is a function of no arguments that, when called, returns the rest of the lazy list (as a lazy list, which looks like either `()` or `(b . g)`). Lazy lists can be finite or infinite. In homework we implemented a version of `filter` for lazy lists.

B. Implement the following function, which is `map` for lazy lists. (By the way, my solution is four nicely formatted, not-overly-long lines, in addition to the five lines given.)

```
;Input f: A function of one argument.
;Input lazy: A lazy list.
;Return: A lazy list. The results of applying f to the elements of lazy.
(define map-lazy
  (lambda (f lazy)
```

```
void myFunc() {
    int myArray[1000000000];
    /* ...do stuff with myArray here... */
}
```

C.A. My C code contains a function of the form above. At run time, this function causes my program to terminate with a segmentation fault. Explain why, using technical terms that we have discussed in this course. Continuing in English, tell me what I should change in `myFunc` to get a version that works. (Changes outside `myFunc` are not allowed.)

C.B. Write the new `myFunc` in C.

The C program below is listed in two columns, for the sake of page space.

```
#include <stdio.h>

typedef struct Animal Animal;
struct Animal {
   int numLegs;
   int numEyes;
};

void attachLeg(Animal *a) {
   a->numLegs += 1;
}

int mystery(int *v) {
   v[4] = v[3];
   return v[2];
}
```

```
void setEyePairs(Animal *a, int v) {
   v = v * 2;
   a->numEyes = v;
}

int main() {
   Animal dog = {4, 2};
   Animal spider = {8, 8};
   attachLeg(dog);
   printf("%d\n", dog->numLegs);
   int v[5] = {9, 2, 1, 6, 4};
   v[3] = mystery(v);
   printf("%d %d %d %d %d\n",
      v[0], v[1], v[2], v[3], v[4]);
   setEyePairs(spider, v[1]);
   printf("%d %d %d %d %d\n",
      v[0], v[1], v[2], v[3], v[4]);
    return 0;
}
```

D.A. There are three syntax errors in the `main` function. Please fix them.

D.B. Once fixed, what does the program print? No explanation is needed.

The code below is written in a language that supports multiple kinds of scoping. The functions `f` and `g` are defined using slightly different syntax, which causes them to be scoped differently. This language's `Module` is essentially identical to Scheme's `let`.

```
x = 3;                          (* this is an example of a comment *)
f[y_] := x + y;
g[y_] = x + y;
Print[f[4]]; Print[g[4]];       (* this line prints 7 and 7 *)
x = 5;
Print[f[4]]; Print[g[4]];       (* this line prints 9 and 7 *)
Module[{x = 6},
    Print[f[4]]; Print[g[4]];]; (* this line prints 9 and 7 *)
```

E.A. In general, what is the distinction between static and dynamic scope?

E.B. Would you guess that `f` above is statically scoped, dynamically scoped, or something else?

E.C. What about `g`?