B Exam May 1996: Programming Languages

Problem 1: Someone designed a safe Scheme implementation that evaluates individual expressions. To provide programmers with some feedback about the safety of the program, he wants to equip the implementation with a type system.
Question 1: Which of the program fragments below can he type in

1. a simple type system with base type `int` and all resulting procedure types;
2. an explicitly polymorphic type system; or
3. a simple type system extended with a construct for introducing recursive types?

If you claim that a fragment can be typed in a simply typed version, supply the missing type specifications and ignore the other type systems (since they are extensions). If a fragment can be typed in the explicit polymorphic system, insert type abstractions and type specifications at the appropriate definitions. If it can be typed in a type system with recursive datatypes, provide the datatype *and* change the code accordingly. If a fragment cannot be typed in *any* type system, provide a reason for this failure for each of the three type systems.
Question 2: Use the examples to explain how advanced typing constructs affect code generation.

Fragment 1: `... (let ((f *** (lambda (x ***) x))) ... (f f) ...) ...`
Fragment 2: `... (let ((f *** (lambda (x ***) (x x))) ... (f f) ...) ...`
Fragment 3: `(let ((f *** (lambda (x ***) (x x))) ... (f f))`
Fragment 4:

```
... (let ((f *** (lambda (x ***) x))) ... (f 5) ...) ...
```

The stars in each fragment indicate where type specifications are missing for simply typed programs; the dots indicate where context or other expressions are missing.

Problem 2: Someone claims that "freeing heap-allocated memory requires a theorem." State the "theorem" that he refers to as clearly as possible. Explain how C++ programmers and Scheme programmers convince themselves that the theorem is true.

Problem 3: True or false: a program that is in continuation-passing style allocates no stack space. If you believe this claim, explain why it is true; if not, show a counter-example.