2010 Paper 7 Question 12

Optimising Compilers

A new programming language follows an "evaluate only on need" strategy. The consequence is, for example, that in a program fragment

```
int x = SomeExpression;
if (SomeBoolean) print x;
if (x == 0) x = SomeOtherExpression;
```

the given (potentially complicated) expression is not evaluated on the line that declares `x`, but only gets evaluated if and when `x` is used, as in the print statement on line 2. On line 3 the value of `x` is certainly needed, so if the expression had not been evaluated earlier it must be there.

Optimising compilation normally improves performance of compiled code based on transformations that are "safe" in some sense. Propose forms of analysis and hence optimisation relevant in this case in the circumstances:

(a) It is safe but undesirable to evaluate an expression even if its value will not subsequently be used, but it must not be evaluated a second time. [8 marks]

(b) It is safe to evaluate an expression repeatedly if is evaluated at all, but if the program would not use the value it must not be computed at all. [7 marks]

(c) Extra executable logic has to be put in the program to ensure that each expression is evaluated exactly once if its value is needed, but not at all otherwise. It is desirable to minimise the amount of this extra logic, preserving semantics exactly. [5 marks]

You need only consider the case of optimisation of a single procedure at a time.