1) Answer each part with a few sentences, a diagram or code sequence (5 points each):

a. Under what circumstances might we choose to implement a tree-building compiler instead of a pass-oriented compiler?

b. What considerations influence the decision to use a jump cascade in implementing Boolean operations?

c. What considerations lead to the choice of a heap as a storage allocation mechanism?

d. What information must be stored in a token for typical programming languages?

e. What is a phrase, and why is it important?

f. Consider a left-recursive grammar with embedded connection points. Explain why the transformation that eliminates left recursion can be guaranteed to leave the invocation sequence of the associated “semantic actions” invariant.

g. Define the inherited, intrinsic and synthesized attribute classes. Can we ever be in doubt as to the classification of a particular attribute? Explain briefly.

2) Consider a tree-building compiler that implements attribute computation by algorithms (one per production of the abstract syntax) consisting only of the following kinds of operation:

- Evaluate an attribute or condition
- Visit a specific child node
- Visit the parent node

Each evaluation operation is defined in some arbitrary applicative language.

a. (5 points) To which class must the attribute grammar defining these computations belong? Explain briefly.

b. (5 points) Why is this an appropriate choice of operations for the attribute computations?

c. (10 points) Sketch an implementation mechanism for the algorithms in sufficient detail to convince me that it will work in all cases. (Do not be concerned with the applicative expressions that evaluate attributes; concentrate your attention on overall execution.)