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

a. A compiler writer must devise an appropriate model of source language concepts by target language concepts. Such a model may have imperfections. Give an example of an imperfection that constrains the set of algorithms that can be constructed.

b. Under what circumstances would you choose to use a source-oriented abstract machine code (like P-code) as the target of a compiler?

c. Under what circumstances would you choose to implement a pass-oriented compiler (as opposed to a tree-building or optimizing compiler)?

d. The text suggests that the center substring of an identifier be used as input to a hash function. In practice, however, the identifier lookup runs faster if the hash function is based upon the entire string. Briefly speculate on the reason for this seeming anomaly.

e. Why is a compiler writer only interested in well-defined attribute grammars?

f. What characteristics of partitioned attribute grammars make them particularly interesting to compiler writers?

g. In an LAG(1) grammar, how does \((X_i, a \rightarrow X_j, b) \in DDP^+\) constrain \(i, j\) and \(a\)?

h. Semantic analysis can be broken into two major subtasks:
   - Name analysis
   - Type analysis

   In general, these two subtasks must be carried out together because the results influence each other. Give examples to support this contention.

i. We argued in class that name analysis was an \(O(n)\) algorithm, where \(n\) is the number of identifiers in the program. Sketch the outline of that argument.

j. Does the argument of (i) break down in the case of Pascal record field identifiers? Briefly defend your answer.

k. Discuss the implementation of the operation \(\text{Align}(x, y)\), which returns the smallest integer not smaller than \(x\) that is also divisible by \(y\).
1. What is the normal form theorem for code generation on machines with stacks of bounded depth? Briefly explain its significance for code generation.

m. Distinguish the terms accessible and addressable. How are accessibility and addressability embodied in the compiler’s data structures?

n. Suppose that Sethi-Ullman numbering is used to estimate register usage for a Pascal compiler. Explicit store operations are inserted into the tree whenever the estimated requirements exceed the number of available registers. Will a register ever have to be freed “unexpectedly”? Explain briefly.

o. State the analogy between operator identification (during semantic analysis) and instruction encoding (during assembly). Be specific!

p. What is the aliasing problem, and how is it related to value numbering?
2) You are given a language definition similar to that for MINILAX, which specifies the semantics denotationally and also provides a concrete grammar. You need to create a compiler, and your only tool is an LL(1) parser generator similar to SYNPUT.

a. (5 points) What transformations of the concrete grammar might you consider? Would these transformations guarantee that you could eventually build a parser using SYNPUT?

b. (10 points) How would you reflect the semantics in the parser design, given the fact that the grammar must be transformed? Why will your approach work?

c. (15 points) What is the role of the “lookahead token” in a parser generated by SYNPUT? Is it correct to say that no error can ever be detected by examining a lookahead token? Relate your answer to the use of lookahead tokens in LR parsers.

3) Uwe Kastens claims that the mechanism for implementing short-circuit evaluation given in the text can be simplified: Instead of inheriting two target locations and a boolean, each boolean expression need inherit only one target location and a boolean. The target location is the label that does not define the fall-through position, and the boolean is the value on which a jump to this location should be made. Thus the pair \((L, true)\) would indicate that the boolean expression should jump to \(L\) if it yields \(true\) and fall through if it yields \(false\).

a. (5 points) Suppose that Kastens’ scheme were used, and a node whose operator was “<” inherited these attributes. Suppose further that the operands of this node synthesized the attributes left and right, which defined the access paths to the operand values. Explain how this information would be used to select VAX instructions to implement the operation. (Ignore any possible operand addressability problems.)

b. (10 points) Suppose that a node whose operator was “and” inherited these attributes. Explain how Kastens’ attributes would be computed for the children of this node.

4) Suppose that LAX integer array variables were added to MINILAX. The declaration and use would take the forms:

\[
\text{identifier ': ' 'array' [ '(' (expression ': ' expression ' || ' ', ') ') ' of' ' type specification }
\]

\[
\text{name '[' (expression ' || ' ', ') ') ' ]'}
\]

Assume that all expressions in an array declaration are evaluated in the context of the enclosing block. An array declaration is permitted in the outer block, but its bound expressions may involve only integer constants. A compiler for this
extended MINILAX on the VAX is desired.

a. (10 points) Show how storage will be allocated, and how variable access will be implemented. (Be sure to discuss both simple variable and array element access.) Specify the additional actions that must take place upon entering and leaving a block.

b. (5 points) What calls will be made on the memory mapping module discussed in class in order to allocate storage for the arrays? Be specific!

c. (5 points) How will the Sethi-Ullman numbering be affected by the extension? Be specific!

d. (15 points) Devise a plan for modifying your compiler to handle this extension. Your plan should indicate which modules will be affected, roughly what changes will be necessary, and the amount of time you expect to spend making the change. Identify any areas that you feel will be particularly difficult to modify.