B1. Compilers (25 points) Answer all questions.

1. (10 points) Show that the following grammar is LALR(1) but not SLR(1).

S -> Xa | bXc | ec |bea
X -> e

2. (15 points)

(a) For a given LR(1) grammar, what kinds of conflicts might be present in an LALR(1) parsing table for that grammar? Explain why they could occur.

(b) How can ambiguous grammars be used in conjunction with LR-parsers, even though every ambiguous grammar fails to be LR?

(c) Suppose you plan to use an automatic LL(1) predictive parser generator, which requires that the grammar be LL(1). Perform two transformations to the following grammar to help it become LL(1). Justify why your transformations help it to become LL(1).

S -> Sa | Tb | e
T -> c | cd | fTe

B2. Compilers (25 points) Answer all questions.

1. You are in charge of the symbol table management, type checking, and final code generator for a compiler. In the questions below, be as specific as possible.

(a) (5 points) For each of these phases, list the information you need to obtain from the language designer in order to design the phase.

(b) (5 points) For each piece of information listed above, explain how the information is used in the design of the phase.

2. Consider the basic block below:

```
t1 = a * a
t2 = a * b
t3 = 2 * t2
t4 = t1 + t3
t5 = b * b
t6 = t4 + t5
```

(a) (7 points) Assuming that the variables are not used after this code segment, compute the liveness and next use information (used for register allocation) for each variable at each statement.

(b) (8 points) Rewrite the code with registers assigned, attempting to use as few registers as possible, but not reordering the code.

B3. Compilers (25 points) Answer all questions.

1. The objective of implicit memory deallocation (commonly called garbage collection) is to automatically reclaim the set of memory chunks that will no longer be used by the program.

(a) (4 points) Since the set of memory chunks used by the program cannot be predicted exactly in advance of execution, what approximations are usually used by automatic methods?

(b) (6 points) Describe in detail either a one-shot collector method or an on-the-fly method.

(c) (5 points) Compare and contrast the advantages and disadvantages of a one-shot versus an on-the-fly method for garbage collection.

(d) (3 points) Name some properties of garbage collection techniques that make concurrent garbage collection very hard.

2. (7 points) For the following data flow equations, answer the following questions:

$IN[b] = \cap OUT[P]$, P is predecessor of b
$OUT[b] = (IN[b] - KILL[b]) \cup GEN[b]$

(a) Is this a forward or backward data flow problem?
(b) Is this an intersection or union problem?
(c) Name one data flow problem which is modeled by these equations.
(d) Explain in English what the equations say, without just reiterating the equations.

B4. Compilers (25 points) Answer all questions.

1. (3 points) For each phase below, identify an error that can be detected by that phase, but not detected by an earlier stage of compilation.
(lexical analysis, parsing, semantic analysis, runtime)

2. (4 points) For a language with static scoping, describe the contents of a typical activation record, including the purpose and implementation of each component of the activation record.

3. (4 points) Explain why languages with recursion need a runtime stack in addition to static store.

4. (4 points) Explain how history-sensitive local variables of functions (local variables that maintain their values from one call to the next of the same function) can be implemented for a language that has recursion.

5. (3 points) Name 3 language features that can not be implemented via a runtime stack, but require storage on the heap. Justify why these structures require the heap.

6. (4 points) In object-oriented languages, it is not always possible to determine the type of the receiver at compile time. Explain how method calls can be implemented under this situation.

7. (3 points) What kinds of program analysis would a compiler for an object-oriented language perform in order to generate high quality code given the features of inheritance, polymorphism, and dynamic binding.