B1. Compilers (25 points)

Complete a, b, c, d and e.

a) [10 pts] Suppose that in some programming language an identifier is any string of characters over the alphabet {a, b, c, ..., z, 0, 1, 2, ..., 9, _} (that last character is an underscore), that satisfies these three conditions:

- the string starts with an alphabetic character
- the string ends with an alphabetic character or a digit
- no two consecutive characters are underscores

Thus, a_b_8_9 is an identifier, but a_ _ 7, 6abc, and xyz_ are not identifiers.

i) Give a regular expression for the identifiers in this language.

ii) Give a transition diagram for recognizing identifiers in this language.

b) [5 pts] Most lexical analyzers have a string table associated with them. Describe a data structure for string tables that allows lookup to be done in less than O(n) time, where n is the size of the table. Be sure to state any assumptions that you are making about the language in justifying the appropriateness of your structure.

c) [4 pts] Describe two different varieties of error recovery in lexcial analyzers. You should give a specific example that illustrates the differences in the two methods.

d) [2 pts] When using a tool such as lex that "generates" a lexical analyzer, you still have to provide a regular expression for each token, as well as a complete specification of the actions. So, what does the lexical analyzer "generator" provide for you??

e) [4 pts] Compilers can recognize reserved words during lexical analysis by using a transition diagram for each keyword. Explain why this is sometimes considered to be a bad idea, and what alternative method can be used to recognize reserved words.

B2. Compilers (25 points)

Complete a, b and c.

a) [13 pts] Construct the relevant sets of items and the LALR(1) parsing table (action part only) for the following grammar. Please write very clearly.

```
<block> ::= BEGIN <idlist>  END

<idlist> ::= ID  |  <idlist> ;  ID
```

Here, BEGIN, END, ID and ; are terminals, and <block> and <idlist> are nonterminals.

b) [6 pts] In general, what are the fundamental differences between SLR sets of items and LALR sets of items?

c) [6 pts] Is the grammar given in part a an LL(1) grammar? If not, rewrite the rules of the grammar so that it is LL(1).

B3. Compilers (25 points)

Complete a, b, c, d, e and f.

a) [4 pts] What is peephole optimization and where is it used? Describe two such optimizations.

b) [4 pts] What is a display and where is it used? Give an example.

c) [4 pts] What is a register interference graph and where is it used? Give an example.

d) [4 pts] What is an abstract syntax tree and where is it used? Give an example.

e) [4 pts] What is a heap and where is it used? What additional considerations arise with a heap that do not arise with a stack?

f) [5 pts] What is bootstrapping? What advantage is there to implementing a compiler by means of a sequence of bootstrapping steps instead of just one bootstrapping step?

B4. Compilers (25 points)

Complete a, b and c.

Consider the following basic block in normal form:

```
(1)   t1 := a + b
(2)   t2 := c + d
(3)   t3 := t1 + e
(4)   t4 := t2 + t2
(5)   t5 := t3 + t4
```

a) [8 pts] For each statement, compute the liveness and next-use information that is needed to generate code.

b) [8 pts] Define a simple **getreg** function that will be used to determine where the result of an assignment statement is stored. That is, give a general set of rules for deciding which register or memory location to use as operands in each instruction.

c) [9 pts] Using the **getreg** function that you have defined, generate code for the above basic block using MOV and ADD instructions. Here, both MOV and ADD take two arguments, which can be either registers or memory locations in any combination.