B1. Compilers (25 points)

1. (a) (3 points) Restate the formal rule of inference below in English sentences.

   Let S be the function representing the symbol table environment mapping identifiers to types.

   $$\frac{S \vdash e1 : Int \qquad S \vdash e2 : Int}{S \vdash e1 * e2 : Int}$$

   (b) (7 points) Give a detailed description of the implementation of a type checker, assuming that the language has no polymorphism. Be sure to explain how the whole program gets type checked, and the general method in which type checking rules would be implemented for a single statement.

   (c) (3 points) Explain how coercion is handled in the type checking system.

2. There are many forms of intermediate code, and in fact, many compilers use several kinds within the same compiler, at different phases of compilation.

   (a) (6 points) For the expression, a = b + c * d – 10 + c * d, show a parse tree, dag, postfix, and three-address code representations. Assume that * is higher precedence than + and – which are the same precedence. All operators are left associative.

   (b) (6 points) Discuss the pros and cons of each of these representations, and when they are most appropriate.

B2. Compilers (25 points)

You are building a code generator and concerned not only about correctness but also about generating efficient code for your target architecture.

1. (7 points) For the intermediate code statement, a = b + c, (a, b, and c are user-defined variables) describe an algorithm for generating the final machine code with the goals of both correctness and efficiency. Be sure your code generator exploits the results of generating code from previous instructions.

2. (8 points) Suppose you want to include a separate optimization pass over your intermediate code, before actually generating the final code. Describe 2 peephole optimizations, one local optimization, and one global optimization you would include. Show example code segments from before and after each transformation.

3. (5 points) The terms "optimizer" and "optimized code" are really misleading. Explain both theoretically why these terms are misleading, and demonstrate how aliasing, procedure calls, and flow of control affect the optimization process.

4. (5 points) Because programs are now mostly written in a modular fashion as a set of functions/methods rather than one large monolithic routine, there is a desire to optimize the whole program, rather than each function in isolation. Analysis is performed on the set of all functions in the program, by building a call graph and performing static analysis based on the calling relations represented in the call graph. Describe some challenges that modern programming languages and environments (like C++ and Java, for instance) pose for whole program analysis.

B3. Compilers (25 points)

1. (5 points) Using Thompson's construction, build an NFA from the regular expression

   (ab|c*b)*c*

2. (10 points) Let M be the NFA defined by the following transition table:

	a	b	c	ε
0	{1,2}	{2}	{}	{2}
1	{}	{1,2}	{2}	{}
2	{2}	{}	{0}	{1}

   The start state is 0, and the accepting state is 2.
   Convert M to a DFA. Show your work.

3. (10 points) Convert the following DFA to a minimum state DFA that accepts the same strings. The start state is 0, and the accepting states are 8 and 9. Show your work.

	a	b	c
0	1	5	2
1	6	3	4
2	7	2	6
3	6	8	9
4	2	9	8
5	6	3	3
6	2	7	6
7	2	2	6
8	5	1	2
9	1	1	7

B4. Compilers (25 points)

Consider the following grammar:

S -> A b
A -> BBS | a
B -> b | c
where S is the start symbol.

1. (6 points) Compute the FIRST and FOLLOW sets for all of the nonterminals in the grammar.

2. (13 points) Construct an SLR parsing table for this grammar. Show your work.

3. (6 points) Assume a language with functions and recursion.

   (a) Give 2 examples of data items that can be stored as part of the runtime stack, and explain why they can be stored on the stack.

   (b) Give 2 examples of data items that must be stored on the heap, and explain why they cannot be stored on the stack and maintain correctness.