B1. Compilers (25 points) Answer all questions.

1. (15 points) Given the expression

badhabits := (tv + study + tv) * nightly + games * nightly ;

(a) (2 pts) Show the stream generated by the scanner and passed on to the parser.
(b) (2 pts) Show the string table contents.
(c) (3 pts) Show an abstract syntax tree produced by the parser.
(d) (4 pts) State 2 actions the semantic analyzer would perform with this expression.
(e) (4 pts) State 2 issues the code generator would have to address to deal with this expression.

2. (10 points) Consider the following flex-like specification. The alphabet is the set {a,b,c}. Parentheses are used to show the association of operations and are not part of the input alphabet.

bb*	{return token1;}
c(a|b)*	{return token2;}
ab*c	{return token3;}
caa*	{return token4;}
b*aa*(c|ε)	{return token5;}

(a) Show how the following string would be partitioned into tokens. Label each lexeme with the integer of the correct token class. Assume flex semantics for this question.

caabbbcbbaccacababababbac

(b) Can any of these rules be removed from the specification without changing the scanner's behavior on any string? If no, why not? If so, which rules can be removed and why?

B2. Compilers (25 points) Answer all questions.

1. (7 points) Show that the following grammar is LR(1).

   S -> Aa | bAc | Bc | bBa
   A -> d
   B -> d

2. (6 points) Show that the grammar in question 1 is NOT LALR(1).

3. (12 points) For each of the following language features, describe the possible issue in type checking with this feature, and then an approach for type checking in its presence.

   a. coercion
   b. operator overloading
   c. inheritance
   d. polymorphic functions/methods

B3. Compilers (25 points) Answer all questions.

1. (9 points)
   a. (3 points) What are the two common problems that cause grammars to be non-LL(1)?
   b. (3 points) Show a simple example for each problem.
   c. (3 points) For each problem, what are the symptoms if a top down parser is applied to a grammar with that problem?

2. (3 points) Name and justify 2 language features that require heap storage allocation.

3. (13 points) Consider the following object-oriented program segment written in an object-oriented language.

```
Class Z {
        A: Int;
        B: Int;
        C: Int;
        N(): Int {B<- 10;};
        G(j: Int) : Int { A <- B + C;};
};
Class W inherits Z {
        D: Int;
        E: Int;
        G(k: Int): Int {E <- E + D;};
        H(): Int {B <- B + C + D;};
};
Class Y inherits W {
        F: Int;
        R: Int;
        M(): Int { F <- 4;};
        H(): Int {G <- G + F;};
};
```

   a. (3 points) Draw an inheritance graph for this code segment.

   b. (3 points) Explain the role that the inheritance graph plays during semantic analysis, and then in code generation. Give examples of specific situations of how it is used.

   c. (7 points) Draw a picture of the storage layout for 2 objects of type Z, one object of type W, and one object of type Y, along with the contents of the dispatch tables for this code segment. Assume that the code for a given method N in class Z is located at the address labeled by Z_N:.

B4. Compilers (25 points) Answer all questions.

1. (3 points) Give an example to demonstrate the difference between structural and name equivalence.

2. (6 points) Briefly discuss the relative costs and tradeoffs of the reference counting and mark-sweep garbage collection methods. Be sure to justify why you think there have been so many more garbage collection methods developed since these two methods were first introduced.

3. (12 points) Consider the following programming language statement in an object-oriented language in which this is syntax for a method call.

```
u.G(p,q);
```

a. (6 points) Describe what needs to be done during the semantic analysis phase to enable the code generator to generate code as well as be assured that it is only generating code for statically correct statements.

b. (6 points) Describe what the code generator needs to do to generate correct code for this statement.

4. (4 points) Explain the concept of "live variables" and its use in register allocation algorithms.