Instructions: This is an open book, open note, 110 minute exam. You are not allowed to use your homework assignments or your project. There are 3 questions worth a total of 100 points. You must answer all of the questions. Put your name on all pieces of paper that you turn in. Show all of your work, write legibly and good luck.

1) (50 points) You are to add a new language feature to COOL and enhance your compiler appropriately. Currently has a type-case statement, you are to add a value case statement. This should work for any builtin type.

```
vcase(e) is
  val1 : e1;
  ...
  otherwise : ek;
esacv;
```

Alternatives consist of a literal value, e.g., val1, and an expression, e.g., e1, to be evaluated if the tested expression, e, has that value.

Semantically, this is equivalent to a series of nested if-then-else's with conditions that correspond to expression/literal comparisons. Thus, when a case's statement executes the next statement to be executed is the one following the switch statement itself. The literals must be unique and the ranges must cover the domain of the expression (e.g., all integers), so a default case is (usually) required to handle any values not matched by an explicit case. There are obviously type consistency requirements for the literals and expressions as well as typeing the result of the vcase expression as a whole.

Describe in detail the changes you would make to your compiler to:

a) **(5 pts)** Enhance the scanner to handle the new lexical elements.

b) **(10 pts)** Enhance the parser to handle the new syntactic elements. Be sure to describe any changes to the structure of the AST.

c) **(10 pts)** Give a type inference rule for the vcase expression. You can base your work on the conditional type inference rule.

d) **(10 pts)** Enhance the semantic analysis to handle the new semantic elements. Be sure to handle any error detection necessary.

e) **(10 pts)** Give an operational semantics rule for the vcase expression. You can base your work on the conditional operational semantics rule.

f) **(5 pts)** Enhance the code generator to handle the new AST nodes.

If you need any assumptions, e.g., about string comparisons, just state them explicitly and proceed.

2) (20 points) The question is about the architecture of compilers and of your compiler in particular.

a) **(5 pts)** Sketch the main components of your compiler and the information passed between those components. A box and arrow diagram with accompanying english descriptions is fine.

b) **(2 pts)** Define *compiler phase* and *compiler pass*.

c) **(4 pts)** Identify the phases and passes in your project compiler.

d) **(4 pts)** Could you re-design your compiler to reduce the number of passes? If so, how? If not, why not?

e) **(5 pts)** Select one of the optimizations we have discussed and describe how it would be integrated into your compiler's architecture. Be sure to describe wether it requires a new phase, a new pass, or modification of a phase or pass.

3) (30 points) In class we discussed right linear grammars, also called regular grammars. They have a pretty simple form with rules that look like:

$$\begin{array}{lcl} \text{<A>} & \rightarrow & \text{a <B>} \\ & | & \text{a} \end{array}$$

For the regular expression `(a|b)*cc`, defined over an alphabet of symbols {a,b,c}.

a) **(7 pts)** Give a regular grammar for this expression

b) **(7 pts)** Compute FIRST, FOLLOW for this grammar

c) **(8 pts)** Build the predictive parse table.

d) **(8 pts)** Illustrate the action of your predictive parser for the strings "cc" and "abbcc".