Instructions: This is an open book, open note, 110 minute exam. 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) The JOOS language and compiler distinguish statements from expressions. Some languages, however, consider them to be the same. Actually JOOS has some examples of statement-expressions, e.g., assignments, method invocation and class instance creation. If the values of these are not "consumed" by some enclosing expression then we consider them to be statements. In this question you are to consider a modification to JOOS that turns "if" statements into expressions. For example, we could write:

```
int x,y,z; // assume these are locals

if (x==0)
  z = if (y==0) 0 else 1;
```

The nested "if" is producing a value on the right-hand side of the assignment to `z`. For the purpose of this question, assume that an "if" statement must have conformant types for all of its branches and that the "if"'s type is the most general of those. Note you can think of "void" as a type that only confirms with "void".

Please give the reasoning behind your answers. Describe in detail the changes you would make to your compiler to achieve the following:

a) **(10 pts)** Enhance the scanner and parser to handle any new lexical and syntactic processing that is needed.

b) **(10 pts)** Sketch the new type rules for the "if" expression and describe the type checking code needed to handle your new "if" AST nodes.

c) **(10 pts)** Enhance the code generation patterns to produce code for the new "if" AST.

d) **(10 pts)** Adapt existing peephole patterns to handle the use of "if"s as standalone statements. Don't forget to argue about correctness and convergence of your pattern.

e) **(10 pts)** Show the unoptimized and optimized bytecodes for the code fragment shown above.

Only include code-based solutions after you have provided a high-level description of the changes. Description without code is better than code without description, both code and description will maximize your credit.

2) (20 points) Consider the following WIG program:

```
service {
  global x: int = 87;

  document Add {
    How much do you want to add?
    <input name="addition" type="text" size=4>
  }

  document Total { The total is now <var name="total">.  }

  session Contribute {
     local i: int;
     local x: int;
     x := 7;
     show Add[addition->i];
     take x { x:=x+i };
     show Total[total<-x]
  } ends with "http://www.cis.ksu.edu"

  session ShowIt {
     show Total[total<-(x+x)/2]
  } ends with "http://www.cis.ksu.edu"
}
```

In answering the following questions be sure to make it clear what, if any, assumptions about WIG you are making, e.g., do local names shadow globals?. Be neat in drawing the pictures; if I can't read the diagram I'll assume its wrong.

a) **(10 pts)** Draw a picture of the AST for this program.

b) **(10 pts)** Draw a picture of the symbol table at the end of the "Contribute" session including the linkages between the AST and symbol table.

3) (30 points) This question is about dealing with one common problem in defining parser descriptions for use with modern LR parser generators (e.g., bison) : **conflicts**.

a) **(15 pts)** Describe shift-reduce parsing and how the two kinds of parsing conflicts arise.

b) **(15 pts)** Give a common source of conflicts in expression grammars and describe two solutions that eliminate conflicts. Illustrate the source of conflict and one of the solutions with an example.