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 JOOS and enhance your compiler appropriately. The language currently supports integer and boolean builtin types, you are to add a "float" type for floating point numbers.

```
float val;
val = 2 + 3.5;
```

This should work like any other builtin type, for example you can print it via a call to "tostring". Integers should be automatically promoted to floats when they appear in an expression with a float operand, e.g., the add operator above; The JVM has a bytecode for doing this called "i2f" which pops, converts, and pushes the result. It is legal to cast a float as an integer to round it down; the "f2i" bytecode does this. The arithmetic operators, constant loading, stack load/store, and return bytecodes are the same as for integers except they have a leading "f", e.g., "fadd", "fconst", "fload", "freturn". Unfortunately, all of the nice conditional branch instructions, such as "if_icmpeq", are not available for floats. Instead there are two bytecodes "fcmpl" and "fcmpg" which perform less-than and greater-than comparisons respectively and push a boolean result on the stack.

Make any reasonable assumptions that you think are necessary, e.g., a float fits in a single stack slot, but, be sure to explicitly state your assumptions.

Please give the reasoning behind your answers. Describe in detail, you can sketch out code if you like, the changes you would make to your compiler to:

a) **(10 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)** Sketch out informal type rules for floats. We covered some of these in lecture.

d) **(10 pts)** Enhance the semantic analysis to handle the new semantic elements.

e) **(10 pts)** Enhance the code generator to handle the new AST nodes. Note that floating point comparisons appearing in conditionals can be treated as syntactic sugar by introducing temporary boolean values to hold the comparison result and then testing the boolean.

2) (20 points) This question is about how one might intelligently integrate garbage collection features into a compiler for Pascal (or some similar language). Assume that you have the following operators: `new(type)` that returns a reference to a heap allocated instance of that `type` and `free(ptr)` that reclaims the heap storage pointed to by `ptr`. The idea is that you take in a program that is without calls to `free` and then you generate code for the program that makes calls to `free` at appropriate places. You can view this as just a source program transformation (adding some bookkeeping data, processing and calls to `free`).

In answering this question you will need to think about the following example:

```
record rType1 is
  Integer x;
end rType1;

rType1 r;

procedure p()
 pointer to rType1 pr;
begin
 pr := address of r;

 pr := new(rType1);

 pr := null;

 ...
```

State any assumptions you make about the language.

a) **(15 pts)** Describe how you would implement a reference-counting garbage collection scheme in your compiler? Illustrate the effect of your scheme on the example above.

b) **(5 pts)** Reference-counting can be inefficient in space (to hold the count) and time (overhead for updating counts). It is possible to optimize this situation using static analyses to determine information about values of pointers or expresions at certain program points. What kind of information would be useful for this problem?

3) (30 points) This question is about dealing with two common problematic features in defining parsers for modern programming languages: precedence and ambiguity.

a) **(15 pts)** Describe what precedence means? In the context of expression parsing discuss two strategies for implementing operator precedence? Give an example of an expression grammar and apply the strategies.

b) **(15 pts)** Describe what ambiguity means? Sketch the simplest grammar fragment that handles both *if-then* and *if-then-else* statements. Illustrate the ambiguity. How does one fix this problem?