CIS (4|5)61
Winter 2017 Final Exam

Your name: _____

Read through the whole exam once before you start working on answers, so that you see how the questions are related.

Some answers require diagrams. Work them out on scratch paper before writing in the exam. I may deduct points or not give any credit for writing or diagrams that are too difficult to read.

Think, then draft, then write. No other order works.

Total: _____

1. Consider the following grammar. (I use λ as a place-holder for an empty production.)

$\langle S \rangle$ ::= $\langle Tree \rangle$ \$

$\langle Tree \rangle$ ::= $\langle Leaf \rangle$

$\langle Tree \rangle$ ::= $\langle Head \rangle$ `:` $\langle Children \rangle$ `;`

$\langle Head \rangle$ ::= v

$\langle Children \rangle$ ::= $\langle Children \rangle$ $\langle Tree \rangle$

$\langle Children \rangle$ ::= λ

$\langle Leaf \rangle$ ::= v

Part 1 [5 points]: Assuming the strings *a, b, c, ...* all match the lexical pattern for the token v, which of these sentences are accepted by the grammar? (Circle accepted sentences.)

- `a: ;`
- `a b c`
- `a: b c;`
- `a: b c d: e f ; g h;`
- `a: b: c: ; ; ;`

(score)

Part 2 [10 points]: Here's that grammar again. Show a parsing state in which LALR(1) lookahead resolves an LR(0) conflict.

$\langle S \rangle$ ::= $\langle Tree \rangle$ \$

$\langle Tree \rangle$ ::= $\langle Leaf \rangle$

$\langle Tree \rangle$ ::= $\langle Head \rangle$ `:` $\langle Children \rangle$ `;`

$\langle Head \rangle$ ::= v

$\langle Children \rangle$ ::= $\langle Children \rangle$ $\langle Tree \rangle$

$\langle Children \rangle$ ::= λ

$\langle Leaf \rangle$ ::= v

(score)

2. [5 points] Consider the following Quack program.

```
class Weight(w: Int) {
  this.w = w;
  def inc(delta: Int): Weight {
     return Weight(this.w + delta);
  }
}

class Height(h: Int) {
  this.h = h;
  def inc(delta: Int): Height {
     return Height(this.h + delta);
  }
}

x = 42;
if x > 13 {
   measure = Weight(x);
} else {
   measure = Height(x);
}
size = measure.inc(13);
```

Will this program pass the type checker? If not, why not? If so, what value will be computed for `size`?

3. [5 points] Consider the following Quack program.

```
class RGB(r: Int, g: Int, b: Int) {
   this.r = r; this.g = g; this.b = b;
   def STR(): String {
        return "rgb(" + this.r.STR() +
                "," + this.g.STR() +
                 "," + this.b.STR() +
               ")";
   }
}

class Red() extends RGB {
     this.r = 255; this.g = 0; this.b = 0;
     def STR(): String { return "RED"; }
}

class Purple() extends RGB {
       this.r = 150; this.g = 0; this.b = 150;
       def STR(): String { return "PURPLE"; }
}

color = RGB(0,0,0);
color.PRINT();
"\n".PRINT();
color = Red();
color.PRINT();
"\n".PRINT();
```

What will this Quack program print?

(score)

4. [5 points] While the Quack type system enforces contravariance for most arguments to methods that override a superclass method, the implicit "this" argument (also known as the "receiver" object) is not contravariant. Why is it ok for the implicit "this" argument to be a subtype of the "this" argument of the method from the superclass?

5. [5 points] Our compiled programs use three areas of program storage:

- *text* area: Program code goes here.
- *stack* area: Local variables and some other dynamic information such as method return addresses go here.
- *heap* area: We put objects here. Class method tables can go here or in the *text* area.

Allocating storage in the *stack* area is more efficient than allocating storage in the *heap* area. Why don't we put objects in the *stack*? What could go wrong if we did?

(score)

6. [10 points] I made a mistake in the design of the Quack language. I intended that the == operation as syntactic sugar would make it easy to create user-defined EQUALS methods for each class, or to inherit the EQUALS method of the Obj class. But the following program is rejected by the Quack compiler:

```
class Pt(x: Int, y: Int) {
   this.x = x;
   this.y = y;

   /* Override Obj.EQUALS */
   def EQUALS(other: Pt): Boolean {
       return this.x == other.x and
              this.y == other.y;
   }
}
```

Explain why this class, which overrides the EQUALS method of the Obj class, must be rejected. Be as explicit and concrete as possible. Don't just tell me that it breaks the covariance/contravariance rule of the Quack type system; explain how the object code created for this class could crash if we compiled it the way we compile other Quack classes. If possible, illustrate with a little bit of Quack code that will result in an invalid pointer dereference (typically a 'segfault'), or draw a picture to illustrate the issue.

(Additional space if needed)