There are 6 problems on this exam. It is 11 pages long, so make sure you have the whole exam. You will have 80 minutes in which to work on the problems. You will likely find some problems easier than others; read all problems before beginning to work, and use your time wisely. The examination is worth 80 points total (1 minute/point). The point breakdown for the parts of each problem is printed with the problem. Some of the problems have several parts, so make sure you do all of them.

This is an open-book, open-notes examination. No electronic assistance is allowed.

When writing code, do not worry too much about getting the syntax exactly right. You will not be penalized unless it makes the meaning unclear. Use common sense: I do not expect you to write 100s of lines of code or text; if you find yourself doing so, there is a simpler, shorter answer.

Do all written work on the exam itself. If you are running low on space, write on the back of the exam sheets and be sure to write (OVER) on the front side. It is to your advantage to show your work—we will award partial credit for incorrect solutions that are headed in the right direction. If you feel rushed, try to write a brief statement that captures key ideas relevant to the solution of the problem. If you show your work, clearly indicate what your answer is.

If you finish in the last ten minutes of the exam, please remain in your seat until the end of the exam as a courtesy to your fellow students.

Name ________________________________________________________________

Student ID __________________________________________________________
<table>
<thead>
<tr>
<th>Problem</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>80</td>
<td></td>
</tr>
</tbody>
</table>
1. True/False [16 pts]

(parts a–h; -1 point for each wrong answer, 0 points for each blank answer, 2 point for each correct answer. Therefore, the score for this problem is \( \max(0, 2 \cdot \#\text{correct} - \#\text{incorrect}) \).

a. _____ The following function is tail-recursive:

```scala
def search(target: Int, t: Tree[Int]) = t match {
  case Leaf => false
  case Node(x, l, r) =>
    if (target < x)
      search(target, l)
    else if (target > x)
      search(target, r)
    else true
}
```

b. _____ A closure captures free variables in its environment.

c. _____ With call-by-need semantics, an expression will be evaluated at least as many times as with call-by-value semantics.

d. _____ Reference counting garbage collection cannot collect cycles of garbage.

e. _____ A good module system should separate implementation and specification.

f. _____ Generational garbage collection is based on the assumption that older objects are more likely to be garbage.

g. _____ The parser builds an intermediate representation of the program that is used by the scanner to recognize tokens.

h. _____ With dynamic scoping, variables are bound to locations rather than to values.
2. Scanning [10 pts]

(a) [4 pts] In C, string literals start and end with a double quote ("). They may contain any character except newlines. They may contain a double quote or a backslash (\) only if preceded by a backslash. Write a regular expression for C strings. Use [abc] to denote any one of the characters a, b, or c; use [ˆabc] to denote any character except a, b, or c. Use \n to denote a newline character.

(b) [6 pts] Draw, as a “circle-and-arrows” diagram, an NFA for the above comments. Use “other” to indicate any input character other than the ", \, or \n. Be sure to label the start state and any final, accepting states (use a double circle to indicate an accepting state).
3. Parsing [14 pts]

Consider the following LL(1) grammar for a subset of Lisp:

\[
\begin{align*}
Pgm & \rightarrow Exp \, $ \\
Exp & \rightarrow \text{atom} \\
& \quad | 'Exp \\
& \quad | (Exp \, Exps) \\
Exps & \rightarrow Exp \, Exps \\
& \quad | \varepsilon
\end{align*}
\]

$ is used to indicate end-of-file. ε indicates the empty string. The goal symbol is Pgm.

(a) [6 pts] Consider a recursive-descent parser running on the input \((\text{cdr} \ (a \ b \ c)) \, $\). When the ‘ is matched, what functions of the parser (including the current function) are active on the parser’s call stack?
(b) [8 pts] In a programming language of your choice, sketch how you would implement an AST for this subset of Lisp. If you use an OO language, include the superclass and fields of each class, if any; do not bother with methods or constructors.
4. Scoping [10 pts]

(a) [6 pts] Below is an incomplete program written in a language with nested functions:

```java
def f = {
    var x: Int
    def g = print(x)
    def h = ... 

    x = 1 // assign to x 
    h()
}
```

Fill in the declaration of the function \( h \) so that the program produces different outputs under static and dynamic scoping.

```java
def h =
```

(b) [4 pts] Give an advantage of static scoping over dynamic scoping, and an advantage of dynamic scoping over static scoping.
5. **Modules** [10 pts]

(a) [6 pts] Name three advantages that modules bring to software development. Briefly describe them in one or two sentences each.

(b) [4 pts] In Java, if a field is declared `static`, there is exactly one location for the field in the address space of the process, rather than one location per object as with non-`static` fields. Describe one problem that might be encountered when constructing a large software system where non-final, `static` variables are used frequently.
6. Functional programming [20 pts]

An environment is a mapping from names to values. One way to represent an environment is as a hash table. In Java, this would be: `HashMap<String,Value>;` in Scala: `HashMap[String,Value]`. Environments can also be represented as functions from String to Option[Value]. If an environment `env` maps `x` to value `v`, then the function call `env(x)` returns `Some(v)`. If `x` is not in the environment, then the call returns `None`.

Thus, we can define the `Env` to be another name—an alias—for the function type:

```scala
type Env = String => Option[Value]
```

In each of the following parts, your answer should use only first-class functions and basic control constructs like `if`. Do not use any other data structures. Do not use assignment. You may write helper functions if necessary.

(a) [5 pts] Show how to implement the empty environment. The function `empty` returns the empty environment. Fill in the function body.

```scala
def empty: Env =
```
(b) [5 pts] Show how to implement the lookup operation. The `lookup` function takes an environment and a name and returns an optional value—`Some(v)` if the environment maps the name to `v` and `None` if the environment contains no mapping for `v`. Again, use only first-class functions and basic control constructs.

```python
def lookup(env: Env, x: String) =
```

(c) [5 pts] Show how to implement the insert operation. The `insert` function takes an environment, a name, and a value and returns a new environment that maps the name to the value and contains all other mappings in the original environment.

```python
def insert(env: Env, x: String, v: Value) =
```
(d) [5 pts] Finally, show how to implement the union operation. The `union` function takes two environments and returns a third containing all bindings in both environments. If both environments contain a binding for the same name, then the second environment’s binding has precedence.

```python
def union(env1: Env, env2: Env): Env =
```