2 hours (maximum)
Closed Book

- You may use both sides of one full sheet (8.5x11) of paper covered with any notes you like.
- This exam has 14 pages, including this cover page. Do all your work on these exam sheets.
- Be specific and clear in your answers.
- Show all your work if you wish to be considered for partial credit.

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

Name: __________________________

Email: __________________________

DO NOT TURN TO NEXT PAGE TILL YOU GET PERMISSION
1. [10 points] Memory Layout. For the MiniJava program listed below, show what values are in the stack and the heap at runtime after the return expression for foo(3, 42) has been computed and stored in the return value slot, but before the epilogue for the method foo(3, 42) has been executed. Draw the memory layout on the page 3, which is blank. Show where the frame pointer will be pointing, show where the old `%ra` and `%fp` registers are stored, and show where all of the parameters, locals, and member variables are stored and their current values. For the values of pointers (including `%fp` and `%sp`) draw arrows to indicate what address the pointer variable contains.

class Recursive {
    public static void main(String[] a) {
        System.out.println(new Bar().foo(1, 42));
    }
}

class Bar {
    int sum;

    public int foo(int start, int p) {
        int product;

        if (start < 2) {
            sum = 0;
        } else {
            System.out.println(sum);
        }

        sum = sum + start;

        if (2 < start) {
            product = 1;
        } else {
            product = this.foo(start + 1, p) * start;
        }

        return product;
    }
}
2. [10 points] Memory Layout. For the MiniJava program listed below, show what values are in
the stack and the heap at runtime after the second assignment to the variable junk. Draw the
memory layout on the page 5, which is blank. Show where the frame pointer will be pointing,
show where the old %ra and %fp registers are stored, and show where all of the parameters,
locals, and member variables are stored and their current values. For the values of pointers
(including $fp and $sp) draw arrows to indicate what address the pointer variable contains.

class Array Args {
    public static void main(String[] deNada) {
        System.out.println(new Inner().testing());
    }
}

class Inner {
    public int[] makeArray(int size) {
        return new int[size];
    }

    public int setArrayElt(int[] array, int location, int value) {
        array[location] = value;
        return 1;
    }

    public int testing() {
        int[] a;
        int junk;
        a = this.makeArray(2);
        junk = this.setArrayElt(a, 0, 7);
        junk = this.setArrayElt(a, 1, 1);
        System.out.println(a[0] + 1);
        System.out.println(a[1] + 1);
        return a[1] + a[1] + 1;
    }
}

(page for drawing memory layout for problem #2)
3. [10 points] Memory Layout. For the MiniJava program listed below, show what values are in the stack and the heap at runtime after the return expression for getY() has been computed and stored in the return value slot, but before the epilogue for the method getY() has been executed. Draw the memory layout on the page 7, which is blank. Show where the frame pointer will be pointing, show where the old %ra and %fp registers are stored, and show where all of the parameters, locals, and member variables are stored and their current values. For the values of pointers (including $fp and $sp) draw arrows to indicate what address the pointer variable contains.

```java
class ClassRef {
    public static void main(String[] a) {
        System.out.println(new Foo().testing());
    }
}

class Foo {
    Bar b;
    public int testing() {
        int a;
        b = new Bar();
        a = b.changeY(7);
        return b.getY() + a;
    }
}

class Bar {
    boolean x;
    int y;
    public int changeY(int p) {
        y = p;
        return y;
    }

    public int getY() {
        return y;
    }
}
```
(page for drawing memory layout for problem #3)
4. [15 points] Translating to Java byte code.
For the following two C code snippets, write the Java byte code that implements the snippet. Assume short-circuited logical AND and OR, and that logical AND has higher precedence than logical OR.

```java
do { a = a + 1; } while (a > 10);
```

```java
s && q || (3 < x)
```
5. [15 points] Translating to 3-address code.
For the following two C code snippets, write the 3-address code that implements the snippet. You can assume that each variable can be accessed directly in 3-address code (e.g. \( a = a + 1 \)). Also, you can create as many temporaries as you need (e.g. \( t2 = t1 + 1 \)). Assume short-circuited logical AND and OR, and that logical AND has higher precedence than logical OR.

\[
\text{if ( } y < b \text{ ) } \{ \ a = x + y * z; \ \} \ \text{else } \{ \ p = q; \ \}
\]

\[
\text{s \&\& q } \lor (3 < x) \\ \ \ \ \ \ \ \text{\textbackslash \ put the result in t1}
\]
6. [20 points] LR Parsing Table.

a) For the following grammar:

(1)  \( S \rightarrow L \)
(2)  \( S \rightarrow \epsilon \)
(3)  \( L \rightarrow L \land a \)
(4)  \( L \rightarrow a \)

Draw the LR(1) states and transitions between those states. Then fill in the empty LR parse table on the next page. The parse table contains the number of states you will actually need.
<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$\wedge$</th>
<th>$$$</th>
<th>L</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

b) Is this grammar LR(0)? Why or why not?
7. [10 points] Semantic Analysis in MiniJava.

(a) Label each expression node in the AST on the following page with its type (int, int[], boolean, or class(classname)). Make sure to label each expression node in terms of what its type should be even if its operands have type errors.

(b) On the next page, mark each AST node in which the check types visitor should detect an error or errors. Next to the relevant AST nodes, write one phrase that describes each error. NOTE: Do not worry about possible errors in the part of the AST that is not showing.

(c) On this page, write pseudo-code that more generally implements type checking for the CallExp node. Make sure to indicate whether you are overriding the in, out, or case method. Assume that you can access the kind of type information derived in part (a) of this question by using a HashMap variable called nodeType. For example, nodeType.get(receiver) would return the type of the receiver expression when visiting the CallExp node, nodeType.get(param1) would return the type of the first explicit actual parameter, etc.
8. [10 points] Implementing a new language feature.
Describe how you would extend the MiniJava compiler so that MiniJava includes the equal
to (==) operator. The equal to operator in MiniJava would compare two integers and return
true if those integers have equal value. Briefly describe how each stage of compilation would
be affected.