1. Problem 1 (20 Points)
Languages that employ a reference model of variables also tend to employ automatic garbage collection. Is this more than a coincidence? Explain.

Solution:
Yes, because in reference model there will be copies of the same object. Manually keeping track of all of them (and if there is any references at all) is tedious.

2. Problem 2 (20 Points)
Most binary arithmetic operators are left-associative in most programming languages. However, most compilers are free to evaluate the operands of a binary operator in either order. Are these statements contradictory? Why or why not?

Solution:
Associativity has nothing to do with the order that operands get evaluated (which may even be compiler dependent). Consider the expression 1+(2+3)+(4+5). The fact that which of (2+3) and (4+5) get evaluated first does not have anything with associativity.
3. Problem 3 (30 Points)

Consider the following pseudocode:

```
x : integer – global

procedure set x(n : integer)
    x := n

procedure print x
    write integer(x)

procedure first
    set x(1)
    print x

procedure second
    x : integer
    set x(2)
    print x

main()
    set x(0)
    first()
    print x
    second()
    print x
end
```

What does this program print if the language uses static scoping? What does it print with dynamic scoping? Why?

**Solution:**

static scoping: 1 1 2 2

dynamic scoping: 1 1 2 1

Notice "set x(2)" in procedure "second".
4. Problem 4 (20 Points)

Why don't we use
\[ PP[\text{<program C>}] = CC[C] \circ (\lambda(m,i,o).o) \]
instead of the definition of \( PP[\text{<program C>}] \) on PPT 26, slide 23?

Explain.

**Solution:**

Initial state passed to C cannot possibly contain an error. So error check is unnecessary.

5. Problem 5 (40 Points)

Modify the attribute grammar for Tiny, so that "read" is no longer an intrinsic function, but instead a statement.

Instead of

\[
\text{assign i := read;}
\]

we would have

\[
\text{read(i);}
\]

The read statement would handle exactly ONE identifier.

**Solution:**

Next page:
Tiny's phase-structure grammar:

\[ \text{Remove} \quad \text{Factor} \rightarrow \rightarrow \text{'}read\text{'} \Rightarrow \text{'}read\text{'} \]

\[ \text{and Add} \quad S \rightarrow \rightarrow \text{'}read\text{'} \text{'}(\text{'None'})\text{'} \Rightarrow \text{'}read\text{'} \]

Tiny's AST grammar:

\[ \text{change} \rightarrow \text{'}read\text{'} \]

\[ \text{to} \rightarrow \langle \text{'}read\text{'} \text{'<id:x>'} \rangle \]

Axioms: Remove axioms for \( E \rightarrow \text{'}read\text{'} \)

and Insert new axioms:

\[ E \rightarrow \langle \text{'}read\text{'} \text{'<id:x>'} \rangle \]

\[ \text{code} \uparrow (e) = \text{gen}( \text{gen}( \text{code} \uparrow (e), \text{'}read\text{'} ), \text{'}Save\text{'}, \text{lookup}(x)) \]

\[ \text{next} \uparrow (e) = \text{next} \uparrow (e) + 2 \]

\[ \text{error} \uparrow (e) = \text{if}(\text{lookup}(x) \neq \text{~0}) \text{then} \text{error} \uparrow (e) \]

\[ \text{else} \text{gen}( \text{error} \uparrow (e), \text{'}undefined var in read\text{'} ) \]
6. Problem 6 (40 Points)

Add a 'swap' statement to Tiny. It would look like this:

    swap i :=: j;

Assume a parser ensures i and j are names, not expressions.
Also, assume they are type compatible.

**Solution:**

First, add an item to the C syntactic domain:

    C = ... | <swap I1 I2>

Then, add a case in the definition of CC:

    CC[<swap I1 I2>] = EE[I1] o
    (fn(v1,s1). s1 => EE[I2] =>
    (fn (v2,(m,i,o)).
      let m1 = Replace m  I1 v2 in
      let m2 = Replace m1 I2  v1 in
      (m2,i,o)
    )
  )
  )
7. Problem 7 (30 Points)

Build the functional graph for the binary number 101.11, according to the attribute grammar for binary numbers developed in class.

Solution: