Rebecca Frederick
EECS 345
Written Exercise 1
February 2, 2016

1. <C> → <V> = <D> | <V>

   <V> → x | y | z

   <D> → <E> ? <D> : <D> | <E>

   <E> → <E> || <F> | <F>

   <F> → <F> && <G> | <G>

   <G> → !<G> | <H>

   <H> → (<H>) | <I>

   <I> → true | false

2. Static Semantic Attributes:

type	=	{integer, double}	(synthesized)
typetable(<var>)	=	{integer, double, error}	(inherited)
inittable(<var>)	=	{true, false, error}	(inherited)
typebinding	=	(<var>, {integer, double})	(synthesized)
initialized	=	(<var>, {true, false})	(synthesized)

   Attribute Rules:

<start$_1$> → <stmt$_3$> ; <start$_3$>
<start$_1$>.type := N/A
<start$_1$>.typetable(<var>) := <stmt$_3$>.typetable
<start$_1$>.inittable(<var>) := <stmt$_3$>.initvar
<start$_1$>.typebinding := N/A
<start$_1$>.initialized := N/A

<stmt$_3$>.type := N/A
<stmt$_3$>.typetable := <start$_1$>.typetable
<stmt$_3$>.inittable := <start$_1$>.inittable
<stmt$_3$>.typebinding := N/A
<stmt$_3$>.initialized := N/A
<start$_3$>.type := N/A
<start$_3$>.typetable := <stmt$_3$>.typetable ∪ <start$_1$>.typetable
<start$_3$>.inittable := <stmt$_3$>.inittable ∪ <start$_1$>.inittable
<start$_3$>.typebinding := N/A
<start$_3$>.initialized := N/A
<start$_2$> → <stmt$_4$>
<start$_2$>.type := N/A
<start$_2$>.typetable(<var>) := $\emptyset$
<start$_2$>.inittable(<var>) := $\emptyset$
<start$_2$>.typebinding := N/A
<start$_2$>.initialized := N/A

<stmt$_4$>.type := N/A
<stmt$_4$>.typetable := <start$_2$>.typetable
<stmt$_4$>.inittable := <start$_2$>.inittable

<stmt$_4$>.typebinding := N/A
<stmt$_4$>.initialized := N/A
<start$_3$>.type := N/A
<stmt$_1$> $\rightarrow$ *<declare$_2$>*
<stmt$_1$>.type := N/A
<stmt$_1$>.typetable(<var>) := $\emptyset$(inherited from <start>)
<stmt$_1$>.inittable(<var>) := $\emptyset$(inherited from <start>)
<stmt$_1$>.typebinding := N/A
<stmt$_1$>.initialized := N/A

<stmt$_2$> $\rightarrow$ *<assign$_2$>*
<stmt$_2$>.type := N/A
<stmt$_2$>.typetable(<var>) := $\emptyset$(inherited from <start>)
<stmt$_2$>.inittable(<var>) := $\emptyset$(inherited from <start>)
<stmt$_2$>.typebinding := N/A
<stmt$_2$>.initialized := N/A

<declare$_1$>.type :=
<declare$_1$>.typetable(<var>) :=
<declare$_1$>.inittable(<var>) :=
<declare$_1$>.typebinding :=
<declare$_1$>.initialized :=

<type$_1$>.type :=
<type$_1$>.typetable(<var>) :=
<type$_1$>.inittable(<var>) :=
<type$_1$>.typebinding :=
<type$_1$>.initialized :=

<type$_2$>.type :=
<type$_2$>.typetable(<var>) :=
<type$_2$>.inittable(<var>) :=
<type$_2$>.typebinding :=
<type$_2$>.initialized :=

<assign$_1$>.type :=
<assign$_1$>.typetable(<var>) :=
<assign$_1$>.inittable(<var>) :=
<assign$_1$>.typebinding :=
<assign$_1$>.initialized :=

<expression$_1$>.type :=
<expression$_1$>.typetable(<var>) :=
<expression$_1$>.inittable(<var>) :=
<expression$_1$>.typebinding :=
<expression$_1$>.initialized :=

<expression$_2$>.type :=
<expression$_2$>.typetable(<var>) :=
<expression$_2$>.inittable(<var>) :=
<expression$_2$>.typebinding :=
<expression$_2$>.initialized :=

<value$_1$>.type :=

<value$_1$>.typetable(<var>) :=
<value$_1$>.inittable(<var>) :=
<value$_1$>.typebinding :=
<value$_1$>.initialized :=

<value$_2$>.type :=
<value$_2$>.typetable(<var>) :=
<value$_2$>.inittable(<var>) :=
<value$_2$>.typebinding :=
<value$_2$>.initialized :=

<value$_3$>.type :=
<value$_3$>.typetable(<var>) :=
<value$_3$>.inittable(<var>) :=
<value$_3$>.typebinding :=
<value$_3$>.initialized :=

Table 1: Attribute Rules

3. asdf

4. Loop Invariants:

   Outer (while) loop invariant: The elements $A[bound \ldots n-1]$ are in non-decreasing order

   Inner (for) loop invariant: The elements $A[bound \ldots n-1]$ are in non-decreasing order and $A[0 \ldots i] \leq A[i+1] \leq A[bound-1]$

   Precondition: $n \geq 0$ and A contains n elements indexed from 0

```
bound = n;
while (bound > 0) {
  t = 0;
  for (i = 0; i < bound - 1; i++) {
    if (A[i] > A[i+1]) {
      swap = A[i];
      A[i] = A[i+1];
      A[i+1] = swap;
      t = i + 1;
    }
  }
  bound = t;
}
```

   Postcondition: $A[0] \leq A[1] \leq \cdots \leq A[n-1]$

5. M_{state}(<var> = <expression>, S) =

```
{
    // test that <var> is a legal name in the language
    if M_name(<var>) = 'Error'
        return 'Error'

    // test that <var> has already been declared
    if Lookup(M_name(<var>), S) = 'Error'
        return 'Error'

    // calculate the value of <expression> using the old state
    V = M_value(<expression>, S)
```

```
    if V = 'Error'
        return 'Error'

    // calculate a new state including any side effects from evaluating <expression>
    S_1 = M_state(<expression>, S)

    // remove <var> from the new state
    Remove(M_name(<var>), S)

    // return the new state with the updated value of <var> added
    return Add(M_name(<var>), V, S_1)

}
```

M_{state}(if <condition> then <statement$_1$> else <statement$_2$>, S) =

```
{
    S_1 = M_state(<condition>, S)

    if M_boolean(<condition>, S_1) = true
        return M_state(<statement_1>, S_1)
    else if M_boolean(<condition>, S_1) = false
        return M_state(<statement_2>, S_1)
    else
        return 'Error'
}
```

M_{state}(while <condition> <loop body>, S) =

```
{
    S_1 = M_state(<condition>, S)

    if M_boolean(<condition>, S_1) = true
        return M_state(while <condition> <loop body>, M_state(<condition>, S_1))
    else if M_boolean(<condition>, S_1) = false
        return S_1
    else
        return 'Error'
}
```