Type Checking: Declarations

$$
\begin{array}{lcll}
T & \longrightarrow & \mathsf{int} & \{\ T.\mathit{type} = \mathit{int};\ \} \\
T & \longrightarrow & \mathsf{float} & \{\ T.\mathit{type} = \mathit{float};\ \} \\
D & \longrightarrow & T\ \mathsf{id} & \{\ D.\mathit{type} = T.\mathit{type}; \\
 & & & \mathit{sym\_enter}(\mathsf{id}.\mathit{name},\ D.\mathit{type});\ \} \\
D & \longrightarrow & D_1,\ \mathsf{id} & \{\ D.\mathit{type} = D_1.\mathit{type}; \\
 & & & \mathit{sym\_enter}(\mathsf{id}.\mathit{name},\ D.\mathit{type});\ \} \\
\}
\end{array}
$$

Type Checking Expressions

```
E  ⟶  int_const     { $E.type = int;$ }
E  ⟶  float_const   { $E.type = float;$ }
E  ⟶  id            { $E.type = sym\_lookup(id.name, \mathtt{type});$ }
E  ⟶  $E_1 + E_2$   { if ($E_1.type \notin \{int, float\}$) OR
                              ($E_2.type \notin \{int, float\}$)
                          $E.type = error;$
                      else if $E_1.type == E_2.type == int$
                          $E.type = int;$
                      else $E.type = float;$
                    }
```

Type Checking (Contd.)

```
E  ⟶  $E_1$ [ $E_2$ ]   { if $E_1.type == \mathsf{array}(\mathbf{S}, \mathbf{T})$ AND
                                  $E_2.type == \mathsf{int}$
                              $E.type = \mathbf{T}$
                        else $E.type = \mathsf{error}$ }
E  ⟶  * $E_1$           { if $E_1.type == \mathsf{ptr}(\mathbf{T})$
                              $E.type = \mathbf{T}$
                        else $E.type = \mathsf{error}$ }
E  ⟶  & $E_1$           { $E.type = \mathsf{ptr}(E_1.type)$ }
```

Type Checking (Contd.)

```
E  ⟶  $E_1\ E_2$         { if $E_1.type \equiv \mathsf{arrow}(\mathbf{S}, \mathbf{T})$ AND
                                  $E_2.type \equiv \mathbf{S}$
                              $E.type = \mathbf{T}$
                        else
                              $E.type = \mathsf{error}$ }

E  ⟶  ( $E_1$, $E_2$ )   { $E.type = \mathsf{tuple}(E_1.type, E_2.type)$ }
```

Resolving Names

What entity is represented by `t.area()`?

- Determine the type of `t`.
  `t` has to be of type user(c).
- If c has a method of name `area`, we are done.
  Otherwise, if the superclass of `c` has a method of name `area`, we are done.
  Otherwise, if the superclass of superclass of `c`...

$\Longrightarrow$ Determine the nearest *superclass* of class c that has a method with name `area`.

Resolving Names (Contd.)

```
class Rectangle {
  int x,y; // top lh corner
  int l, w; // length and width

  Rectangle move() {
    x = x + 5;     y = y + 5;
    return this;
  }

  Rectangle move(int dx, int dy) {
    x = x + dx;    y = y + dy;
    return this;
  }
}
```

Resolving Names (Contd.)

What entity is represented by `move` in `r.move(3, 10)`?

- Determine the type C of `r`.
- Determine the nearest *superclass* of class C that has a method with name `move`

  such that `move` ***is a method that takes two*** `int` ***parameters.***

Type Checking Statements

$$
\begin{array}{lcll}
S & \longrightarrow & id \coloneqq E & \{\ \text{if isSubType}(E.type,\ id.type) \\
 & & & \qquad\qquad S.type == \mathsf{void} \\
 & & & \text{else } S.type = \mathsf{error}\ \} \\
S & \longrightarrow & S_1;\ S_2 & \{\ \text{if } (S_1.type == S_2.type == void) \\
 & & & \qquad\qquad S.type == \mathsf{void} \\
 & & & \text{else } S.type = \mathsf{error}\ \} \\
S & \longrightarrow & \mathsf{if}\ E\ \mathsf{then} & \\
 & & \quad S_1\ \mathsf{else}\ S_2 & \{\ \text{if } (S_1.type == S_2.type == void) \\
 & & & \qquad\quad \&\&\ (E.type == bool) \\
 & & & \qquad\qquad S.type == \mathsf{void} \\
 & & & \text{else } S.type = \mathsf{error}\ \}
\end{array}
$$