Class 11
Type Checking
Continued with PA3

Thus, a Simple Type Checker

Type checking `E1 op E2`:

1. `TypeCheck(E1)` return inferred `type(E1)`
2. `TypeCheck(E2)` return inferred `type(E2)`
3. Type rule: Are these what are expected?
   1. `CheckCompatibility(E1, E2)`
   2. `CheckCompatibility(E1,E2,op)`
   3. Emit type errors appropriately
4. `InferType(E1 op E2)`

Type Rules for Function Calls

f is an identifier.
f is a non-member function in scope S.
f has type $(T_1, \ldots, T_n) \rightarrow U$

$$\frac{S \vdash e_i : T_i \text{ for } 1 \leq i \leq n}{S \vdash f(e_1, \ldots, e_n) : U}$$

Where is the type signature?
What the the checks to be done here?
Inference to be done?

Rules for Array Indexing

$$\frac{S \vdash e_1 : T\texttt{[]} \qquad S \vdash e_2 : \texttt{int}}{S \vdash e_1[e_2] : T}$$

How do I read this?
What the the checks to be done here?
Inference to be done?

Rule for Assignment

$$\frac{S \vdash e_1 : T \qquad S \vdash e_2 : T}{S \vdash e_1 = e_2 : T}$$

What the the checks to be done here?
Inference to be done?

If **`Derived`** extends **`Base`**, will this rule work for this code?

```
Base    myBase;
Derived myDerived;

myBase = myDerived;
```

Inheritance Properties

C
|
B
|
A

Animal
|
Mammal
|
Person

Any type is convertible to itself. (**reflexivity**)

If A is convertible to B and B is convertible to C, then A is convertible to C. (**transitivity**)

If A is convertible to B and B is convertible to A, then A and B are the same type. (**antisymmetry**)

This defines a **partial order** over types.

Types and Partial Orders

We say that $A \leq B$ if A is convertible to B.

We have that

$A \leq A$

$A \leq B$ and $B \leq C$ implies $A \leq C$

$A \leq B$ and $B \leq A$ implies $A = B$

Updating the Assignment Rule

$$\frac{S \vdash e_1 : ? \qquad S \vdash e_2 : ?}{S \vdash e_1 = e_2 : ??}$$

How do we fix Type Rules for Function Calls?

f is an identifier.
f is a non-member function in scope S.
f has type $(T_1, \ldots, T_n) \rightarrow U$

$S \vdash e_i : ?$ for $1 \leq i \leq n$

$S \vdash f(e_1, \ldots, e_n) : U$

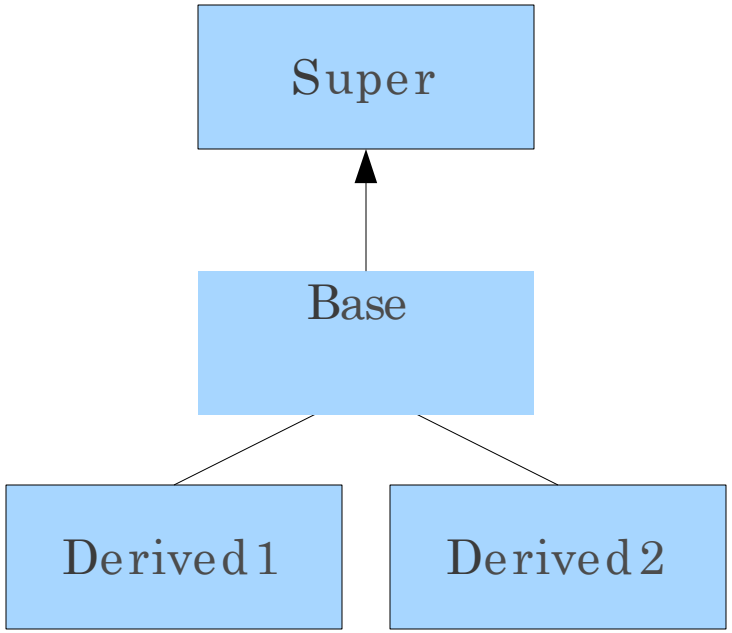
Conditionals

- $S \vdash cond : \texttt{bool}$
  - $S \vdash e_1 : T_1$
  - $S \vdash e_2 : T_2$

WHAT GOES HERE?

- $S \vdash cond \ \texttt{?} \ e_1 \ \texttt{:} \ e_2$: and here??

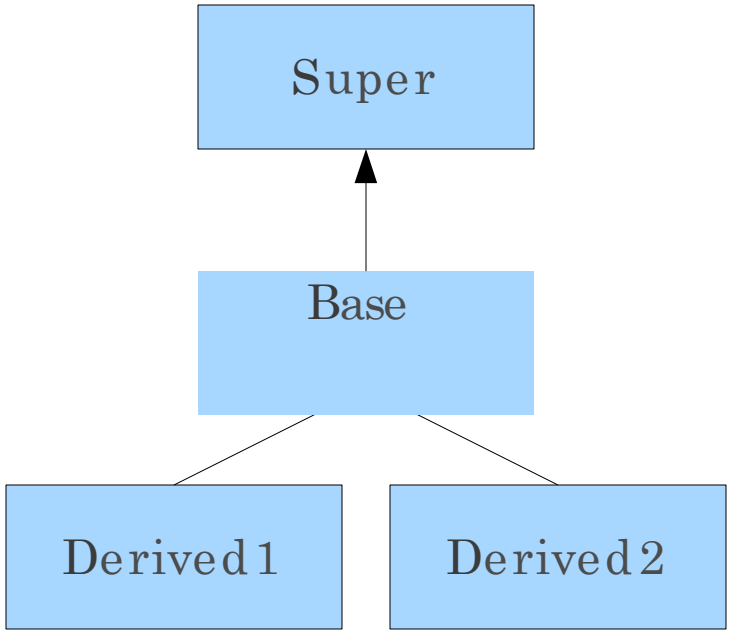
Does this work?



$$\frac{S \vdash e_1 : T_1 \quad S \vdash e_2 : T_2 \quad T_1 \leq T_2 \text{ or } T_2 \leq T_1}{S \vdash cond\ ?\ e_1 : e_2 : \max(T_1, T_2)}$$

```
Base = random()?
           new Derived1 : new Derived2;
```

How about this?



$$\frac{S \vdash e_1 : T_1 \quad S \vdash e_2 : T_2}{S \vdash cond\ ?\ e_1 : e_2 : T} \quad \text{T is lub(T1,T2)}$$

```
Base = random()?
           new Derived1 : new Derived2;
```

Consider type checking this

```
int x, y, z;
if (((x == y) > 5 && x + y < z) || x == z) {
    /* … */
}
```

What happens?

How do we stop from propagating type errors during type checking?

PA3 and the Big Picture

Big Picture

PA2

- Syntax-directed expression "evaluation"
- Syntax-directed code generation

PA3

- Syntax-directed AST creation
- Visitors for creating the dot file for visualization (provided)
- Visitor for checking types
- Visitor for generating code

Later assignments

- Visitor for building a symbol table
- Visitor for allocating memory for variables
- Visitor for doing register allocation

Your Code Structure

In driver, first call the parser to get an AST:

```
mj_ast_parser parser = new mj_ast_parser(lexer);
ast.node.Node ast_root = (ast.node.Node)parser.parse().value;
```

Next create a dot file for the AST for debugging purposes:

```
java.io.PrintStream astout =  new java.io.PrintStream(…);
ast_root.accept(new DotVisitor(new PrintWriter(astout)));
```

Finally, create Type-Checker and an AVRgenVisitor instances:

```
java.io.PrintStream avrsout = new java.io.PrintStream(…);
symtable.SymTable globalST = new symtable.SymTable();
ast_root.accept(new CheckTypes(globalST));
ast_root.accept(new AVRgenVisitor(new PrintWriter(avrsout)));
```