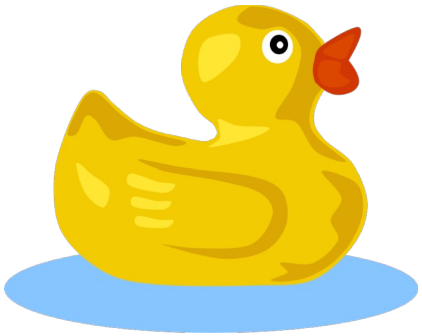
CS 430
Spring 2019

Mike Lam, Professor



$$\text{TEq}\ \frac{\Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1 \text{ `==` } e_2 : \textbf{bool}}$$

Type Checking

Type Checking

- Type system
  - Rules about how data values can be used
- Type checking
  - Act of ensuring that the type system is adhered to
    - Ensure that operands are of compatible types
    - Or of equivalent types if coercions aren't allowed
  - Violations are called type errors
    - Usually, type errors are considered to be bugs
    - Sometimes are reported only as warnings

Type Checking

- Issues to consider:
  - Are declarations explicit or implicit?
  - Which types are equivalent?
  - Are type conversions allowed?
  - Can multiple types be used in some places?
  - When does type checking occur?
  - In general, how pedantic is the process?

Type Checking

- Type declarations
  - Explicit: types required
    - E.g., `int x = 5; float y = 4.2;`
  - Implicit: types not required (or even not allowed)
    - E.g., `x = 5; y = 4.2;`
    - Types are bound at assignment
    - However, these types can often be inferred statically
  - Tradeoff: readability vs. writability and expressiveness

Type Checking

- Type equivalence: name vs. structure
  - Named types vs anonymous types
  - Aliased types (e.g., `typedef` in C)
  - Examples:

```
typedef float celsius;
typedef float fahrenheit;

celsius a = 25.7f;
fahrenheit b;

b = a;  // is this valid?
```

```
typedef struct { int x; } box;
typedef struct { int x; } bin;

box c;
c.x = 5;

bin d;
d = c;      // is this valid?

struct { int x; } e;

e = c;      // what about this?
```

Type Checking

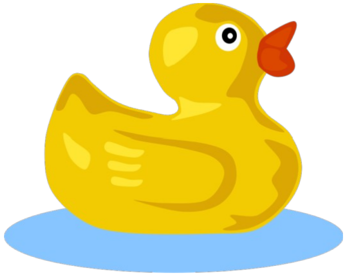
- Type conversions
  - Widening vs. narrowing
    - Latter may cause information loss
  - Implicit vs. explicit
    - Implicit: coercion, e.g., `float x = 5;`
    - Explicit: casting, e.g., `int x = (int)3.14;`

Polymorphism

- Object-oriented inheritance
  - Example of subtypes
- Parameterized functions
  - Uses generic type variables
  - Example: generic list functions in Haskell
    - E.g., `head : [a] → a`
- Abstract data types
  - Models of generic data structure behavior
  - Implementation is hidden from user
  - Can use parameterized types
    - E.g., a `queue<float>` or `queue<int>`
    - Examples: C++ templates and Java generics

Type Checking

- Static vs. dynamic type checking
  - Static: compile time (checked by compiler)
    - E.g., C, Haskell
  - Dynamic: run time (checked by runtime system)
    - E.g., Ruby, Python
    - "Duck typing" is a particular form of dynamic typing
      - If an object has a method, you can call it! ("if it quacks like a duck...")
  - Hybrid: some static, some dynamic
    - E.g., C++, Java
  - Tradeoff: overhead vs. flexibility



Type Checking

- Static type rules are sometimes expressed using proof notation
  - Premises on top, conclusion at the bottom

$$\text{TDec}\,\frac{}{\vdash \texttt{DEC} : \textbf{int}} \qquad \text{TTrue}\,\frac{}{\vdash \texttt{true} : \textbf{bool}} \qquad \text{TLoc}\,\frac{\texttt{ID} : \tau \in \Gamma}{\Gamma \vdash \texttt{ID} : \tau}$$

$$\text{TAdd}\,\frac{\Gamma \vdash e_1 : \textbf{int} \qquad \Gamma \vdash e_2 : \textbf{int}}{\Gamma \vdash e_1 \text{ `+' } e_2 : \textbf{int}} \qquad \text{TEq}\,\frac{\Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1 \text{ `==' } e_2 : \textbf{bool}}$$

$$\text{TFuncCall}\,\frac{\texttt{ID} : (\tau_1, \tau_2, ..., \tau_n) \rightarrow \tau_r \in \Gamma \qquad \Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2 \qquad ... \qquad \Gamma \vdash e_n : \tau_n}{\Gamma \vdash \texttt{ID} \text{ `(' } e_1, e_2, ..., e_n \text{ `)' } : \tau_r}$$

Type Checking

- Strong vs. weak typing
  - Strong typing: all type errors are detected
  - Tradeoff: safety vs. expressiveness
  - Terms often used somewhat loosely
- Evidence of strong typing
  - Static type checking
  - Type inference (even for implicit typing!)
- Evidence of weak typing
  - Dynamic type checking
  - Automatic type conversions
  - Pointer or union types