Type Systems

- Terms to learn about types:
  - Type
  - Type system
  - Statically typed language
  - Dynamically typed language
  - Type error
  - Strongly typed
  - Weakly typed
  - Type safe program
  - Type safe language
- Related concepts:
  - Polymorphism
  - Overloading
  - Coercion

Type – Definition

- Type: a set of values and operations on those values
- Java examples of types:
  - int
    - values = {-2^{31}, ..., -1, 0, 1, 2, ..., 2^{31}-1} (2^{31} = 2,147,483,648)
    - operations = {+, *, ...}
  - boolean
    - values = {false, true}
    - operations = (&&, ||, !, ...)
  - String
    - values = {"", "a", "b", ..., "A", "b", ..., "$", ..., "\Sigma", ..., "aa", "ab", ...}
    - operations = (+, trim(), equals(Object x), clone(), ...)
  - Applet
    - values = {all possible applets}
    - operations = {init(), paint(Graphics g), clone(), ...}

Type System

- A well-defined system of associating types with
  variables and expressions in the language

Statically Typed Languages

- Statically typed. Statically typed means that
  the type of every expression can be determined
  at compile time. Java, ML, and Haskell are
  examples of statically typed languages. (Scheme
  and Ruby are not statically typed though.)
- Each variable has a single type throughout the
  lifetime of that variable at runtime.
- Note that "statically typed" is not the same as
  "type declarations are required".

Dynamically Typed Languages

- Dynamically typed. The types of expressions
  are not known until runtime.
- Example languages: Scheme, Smalltalk, Python, Ruby.
  - Usually, the type of a variable can change dynamically
    during the execution of the program. (Some authors
    make this part of the definition, although we won't for
    341.)
- This is legal Scheme code:
  
  ;; don't need to declare the type of x
  (define x 42)
  (set! x (‘1 2 squid))

Type error

- A type error is a runtime error that occurs when
  we attempt an operation on a value for which that
  operation is not defined.
- Examples:
  
  boolean b, c;
  b = c + 1;
Type Safety

• A program is type safe if it is known to be free of type errors.
  However, the system is allowed to halt at runtime before performing an operation that would result in a type error.
• A language is type safe if all legal programs in that language are type safe.
• Java, Smalltalk, Scheme, Haskell, Ruby, and Ada are examples of type safe languages.
• Fortran and C are examples of languages that are not type safe.
• Some languages for systems programming, for example Mesa, have a safe subset, although the language as a whole is not type safe.

Tradeoffs

• Generally we want languages to be type safe.
• An exception is a language used for some kinds of systems programming, for example writing a garbage collector. The "safe subset" approach is one way to deal with this problem.
• Advantages of static typing:
  – catch errors at compile time
  – machine-checkable documentation
  – potential for improved efficiency
• Advantages of dynamic typing:
  – Flexibility
  – rapid prototyping

Strongly Typed Language

• We’ll define a strongly typed language to be the same as a type safe language.
  However, you may want to avoid this term – or at least explain what you mean. (See the next slide.)
• Weakly typed means "not strongly typed".

Terminology about Types - Problems

• Unfortunately different authors use different definitions for the terms "statically typed" and "strongly typed".
• Statically typed. We define "statically typed" to mean that the compiler can statically assign a correct type to every expression. Others define statically typed to mean that the compiler can statically assign a type to every expression, but that type might be wrong. (By this alternate definition C and Fortran are statically typed.)
• Strongly typed. We’ll define strongly typed to mean the same as type safe. For others, strongly typed implies type safe and statically typed. (Is Scheme strongly typed?)
• To avoid misunderstanding, one can describe a language as e.g. "type safe and statically typed" rather than "strongly typed".
• See the Wikipedia article on "Strongly typed" for a long list of what different people have meant by this term.

Polymorphism, Overloading, and Coercion

• A polymorphic function is a single function that can be applied to several different types. The append function in Haskell is an example: it can be used on lists with any type of element.
  Note that there is just one definition of the function ++.
• An overloaded function is a function with several different definitions. The language implementation chooses the correct definition based on the types of the arguments (and for a few languages based on the type of the result – generally not though). This can be done either at compile time or run time. In Haskell it’s done at compile time. Examples: +, the "show" function.

Polymorphism, Overloading, and Coercion (continued)

• Coercion means that the language implementation converts one type to another to match the requirements of the function or operator. Typical example: coerce an integer to a float in 3 + 4.138
  Try this:
  a = 3 :: Int
  b = 4 :: Float
  c = a + b
• Many languages support this (Fortran, Java, Smalltalk, …)
• However, Haskell does not support coercion. Instead, the type of 3 (for example) is more general than Int or Float – it is (Num t) => t