

## Type Safety

A program is **type safe** if it is known to be free of type errors.

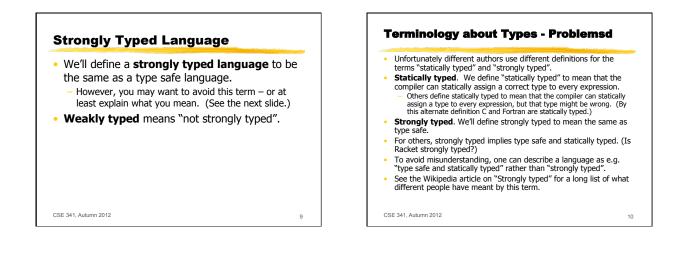
However, the system is allowed to halt at runtime before performing an operation that would result in a type error

- A language is type safe if all legal programs in that
- language are type safe. Java, Smalltalk, Racket, Haskell, Ruby, and Ada are examples of type safe languages.
- Fortran and C are examples of languages that aren't type safe.
- Some languages for systems programming, for example Mesa, have a safe subset, although the language as a whole is not type safe. Also, if you have a language that allows foreign function calls to C functions, then the resulting program is of course not necessarily type safe.

CSE 341, Autumn 2012

## 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. We also often want to support foreign function calls for efficiency.
- Advantages of static typing:
  - catch errors at compile time
  - machine-checkable documentation
  - potential for improved efficiency
- Advantages of dynamic typing: Flexibility
- rapid prototyping
- CSE 341, Autumn 2012



## 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. (++): [a] > [a] [a]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 this done at compile time. Examples: +, the "show" function.

CSE 341, Autumn 2012

11

