Lecture Types 10/13

Types

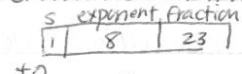
have operations, representations, values.

- Abstract types - defined by the operations on the values
- Concrete types - defined by how their values are represented
- Primitive types - built into the language int
- Constructed types - defined by programmer, using type constructors int[]

- Comparing float. IEEE-754 floating point

Abstractly: operations - binary, unary, comparison, trunc, sqrt, format, e

Concretely:



+0
-1 0≤e≤255 $\rightarrow \pm 2^{e-127} \times 1.f$

$2^{-127} \cdot 1.000/2$ underflow. $a<b$ $b-a == 0$?

$2^{-127} \cdot 0.f$ ← unnormalized (tiny) 0, -0 -0.0f == 0.0f

e=255, f=0 ∞

0.0/0.0 or ∞-∞ Nan (not a number) e=255, f≠0

Nan alt: throw an exception. don't need special vals. can enable trapping

- What are types good for? Debugging (type checking)
  - Abstraction/encapsulation (programs clearer)
  - Specify representation (in low level ops)
  - Compiler can generate faster code (optimization)
  - Annotation (useful for programmer, compiler)
  - Inference (useful for type checking)

- Type Checking
  - strongly typed
    - static Types checked & known before program is run [gen better code, type failures caug
    - dynamic Types are not known until runtime [flexibility, less bureaucratic
  - none

Type Equivalence

- Structural equivalence. Ex. C: typedef int a, typedef int b
- Name equivalence Ex. C struct/class structure

Subtypes

ex: char * ⊆ char const *

(1)

10/15

Lecture 1 ~~Types cont~~

enum color {RGB3} in C is an exposed type.

G is int (1). enumerated types by demand in Java.

Type polymorphism

G a function that accepts ~~types~~ args of varying types

ex overloading a method.

Ex complex operator+ (complex, complex);

a = b+c // b,c of type complex

≡ a = fglooblebaz (b,c)

↑ implementation method → name mangling.

disadvantages: no standards (incompatibilities - Sun, Msft, etc)

names might not be unique

Ex in C double cos (double);

cos(1) converted automatically by compiler.

to; runtime conversion too.

called coercion (implicit type conversion) ad hoc polymorphism

Ex unsigned u = anotherComplicatedfunction(); ← 0

int i = complicatedfunction(); ← -1

if (i < u) // from unsigned/signed comparison $2^{32}-1 < 0$?

print "negative"

solns a) compare w/ a wider int

b) reject @ compile time

c) do two comparisons

Ex long long int = ( ); 64 bit

s	e	f
1	11	52

double d = i;

print(i); print(d);

if (i < d) can fail from squashing int→double.

Lesson: coercions that lose information are dangerous.

Duck Typing - obvious typing (used in Python) runtime type check

Parametric polymorphism - the type of a function isn't fully known at compile time. It has type parameters that are filled in later.

today:
dorm: pick up libr
lotion legs wash face hair fix knee highs paper water
robotics: actions
1+ pages?
navigation
sponsors
color theme?
update method
1x to visit
calendar
banner pic
server hosting?
mission stmt
group descrip
question contact
time lim
web standards
JAC: dvd burning