Type and Abstraction

CS4700

Kenneth Sundberg

Type System

- Type constructors
- Descriptors
- Sometimes called a type algebra

Encapsulation and Information Hiding

- Does the language support thinking at the right level of abstraction
- Does the language support thinking at multiple levels of abstraction
- Consider Float

Type checking

- The act of ensuring that types are compatible
- Coercion is the automatic conversion of one type to another
- Incompatible types that cannot be coerced are type errors

Type Equivalence

- Types are equivalent if no coercion is needed for compatibility
- Name equivalence - types are the same if they are named the same
- Structure equivalence - types are the same if the structure is the same

Strong Types

- Strongly typed languages always detect type errors

Primitive Types

- Integer
- Floating Point
- Complex
- Decimal
- Boolean
- Character

Enumarations

- What forms of coercion are allowed
- What is the scope of the constants
- How visible is the underlying representation

Array

- What are legal subscripts
- Is the array range checked?
- What operations are allowed
  - Slicing
  - Membership
  - Transposition
  - Concatenation

Array Categories

- Static
- Fixed stack dynamic
- fixed heap dynamic
- heap dynamic

Array Implementation

- Are multdimensional arrays allowed?
- Are they row or column major?
- $address[i] = address[0] + i * elementSize$

String Design

- Character array or primitive
- Static or Dynamic length

String operations

- Slice
- Concatenate
- Compare
- Regular Expression Matching

String length

- Static
- Limited dynamic
- Dynamic

Lists

- Very common in functional languages
- List comprehensions
    - Construct one list from another via a function
- Ranges
- Infinite lists
    - If language is lazy infinite lists are possible

Associative Arrays

- Also called hashs, dictionaries, or maps
- Associate a key with a value
- May be balanced trees or hash tables

Records

- Records are formed by concatenating two or more types called fields

Tuples

- Tuples are like records but fields are anonymous
- Arise in generic programming

Unions

- List of types and value is at most one of the listed types
- Discriminated or Free

Pointers

- Entity is a reference to some other
- Pointers are addresses
- References refer to objects

Reference Types

- Reference types are pointers
- Language implementation manages lifetime
- Never null

Pointer Problems

- Dangling Pointer
- Garbage
- Reference Semantics (aliasing)

Pointer Operations

- Pointers refer to memory locations so arithmetic makes sense
- References refer to objects so arithmetic does not make sense
- In both cases dereferencing (which may be automatic) is defined

Garbage Collection

- Reference counting
- Mark and Sweep

Does the language allow user defined types

Are they first or second class citizens

What tools exist for encapsulation and abstraction

User defined type constructor

Critical for generic programming style

Textbook sections covered:

- Section 06-01 (frame 2)
- Section 06-02 (frame 7)
- Section 06-03 (frame 12)
- Section 06-04 (frame 8)
- Section 06-05 (frame 9)
- Section 06-06 (frame 16)
- Section 06-07 (frame 17)
- Section 06-08 (frame 18)
- Section 06-09 (frame 15)

Textbook sections covered:

- Section 06-10 (frame 19)
- Section 06-11 (frame 20)
- Section 06-12 (frame 4)
- Section 06-13 (frame 6)
- Section 06-14 (frame 5)
- Section 11-04 (frame 3)