CS 350: Programming Language Design

Lecture 10

A data type defines a collection of data objects and a set of predefined operations on those objects.

Earlier programming languages offered limited structures.

FORTRAN offered arrays

COBOL offered decimal data and records

ALGOL offered user-defined data types

Type System

* Type checking
  + Ensuring that operands of an operator are compatible
* Program modularization
  + Proper calling of methods and interfaces
* Understanding semantics
  + Expected output

One design issue for all types: What operations are defined and how are they specified

Almost all programming languages provide a set of primitives

* They are those types which are not defined in terms of others

Some primitives are merely reflections of hardware

Primitive: Integer

* A string of bits
  + One of the bits is used for representing the sign
* Different sizes of integers
  + Short, byte, int, long
* Negative values representation
  + Signed magnitude
  + Ones complement
  + Twos complement

Primitive: Floating Point

* Model real numbers, but only as approximations
  + Precision
  + Range
* Languages for scientific use support at least two floating-point types (float and double)
* Usually models hardware exactly, but not typical
* IEE Floating Standard
  + A sign bit followed by several exponent bits and then fractional bits

Primitive: Complex

* Some languages support a complex type
  + C99 and Python
* Each value consists of two floats, the real part and the imaginary part

Primitive: Decimal

* Use in business applications
  + Essential to COBOL
  + C# offers decimal type
* Store a fixed number of decimal digits, in coded form (Binary coded decimal)
  + One digit per byte, or packed two digits per byte
* Operations are done on H/W or simulated in S/W
* Advantage: accuracy
* Weakness: limited range, wastes memory 6 digits, 24 bits

Primitive: Boolean

* Simplest of all
* Range of values: two elements
  + True
  + False
* Advantage
  + Readability over integers used as flags
* Could be implemented as bits, but often as bytes
  + Why? Computers don’t use single bits well, extra info in byte used to identify?

Primitive: Char

* Stored as numeric coding
* Most commonly, ASCII
* An alternative, Unicode
  + An alternative, Unicode
  + Includes characters from most natural languages
    - 16 bit coding
    - 32 bit Unicode
  + Originally used in Java
    - Now supported by many languages
    - Supported by Fortran starting in 2003
* Character String Types
  + Sequences of chars
  + Typical operations
    - Assignment
    - Comparison
    - Catenation
    - Substring reference
    - Pattern Matching
  + Design Issues:
    - Is it a primitive type or just a special array
    - Should the length of strings be static or dynamic
* Character string type in languages
  + C and C++
    - Not Primitive
    - Use char arrays and a library of functions that provide operations
  + Java (and C#, Ruby, and Swift)
    - Primitive via String Class
  + Fortran and Python
    - Primitive type with assignment and several operations
  + Perl, JavaScript, Ruby, and PHP
    - Built-in pattern matching, using regex
* Character String Length Options
  + Static
    - Length can’t be changed after creation
    - Requires no special dynamic allocation
    - Java’s string class -> immutable
  + Limited dynamic
    - Any number of chars
    - Maintain the length or use a special end of a string’s character
    - Require no special dynamic storage allocation
    - C and C++
  + Dynamic length
    - Variable length with no maximum
    - Dynamic storage
      * Must grow and shrink
      * Overhead with allocation and deallocation
    - JS and Perl

String type eval.

* Aids writability
* As a primitive type with static length, they are inexpensive to provide—why not have them
* Dynamic length is nice, but is it worth the expense?

Enumeration

* All possible values, which are named constants are provided in the definition
* Design issues
  + Is enumeration constant allowed to appear in more than one type definition and if so, how is the type of an occurrence of that constant checked?
  + Are enumeration values coerced to integer?
  + Any other type coerced to an enumeration type?
* Aids readability, you don’t need to code values as integer numbers
* Aids reliability
  + Compiler can check
    - Operations
    - No enumeration variable can be assigned a value outside its defined range
  + In C#, F#, Swift, and Java 5.0, enumeration type variables
    - Not coerced into integer types
    - Can’t be assigned a value outside the predefined range