CSE 325/CSE 425: Concepts of Programming Language

Data Types

Dr. Kamruddin Nur

Adjunct Associate Professor, NSU

kamruddin.nur@northsouth.edu

January, 2018

Contents

Data Types

- A **data type** defines a collection of **data** objects and a set of **predefined operations** on those objects
- A **descriptor** is the collection of the attributes of a variable
- Data types can be **primitive** or **user-defined** depending on the language
- An *object* represents an instance of a user-defined (abstract data) type
- **Design issue for all data types:** What operations should be defined and specified?

Primitive Data Types: Integer

- **Almost all** programming languages provide a set of **primitive data types**
- Primitive data types: Those not defined in terms of other data types
- Some primitive data types are merely **reflections of the hardware**
- Others require only a **little non-hardware support** for their implementation

Primitive Data Types: Integer

- Almost always an **exact reflection of the hardware** so the **mapping is trivial**
- There may be as many as eight (8) different integer types in a language
- Java's signed integral data types: **byte**, **short**, **int**, **long**

Data Types in Java

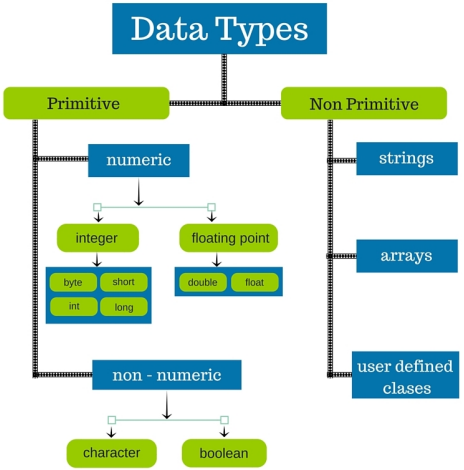

Figure 1: Primitive and Non-primitive Java Data Types

Primitive Data Types: Floating Point

- Model real numbers, but only as approximations
- Languages for scientific use support at least two floating-point types (e.g., **float** and **double**; sometimes more
- Usually exactly like the hardware, but not always
- IEEE Floating-Point Standard 754

Primitive Data Types: Complex

- Some languages support a complex type, e.g., **C99**, **Fortran**, and **Python**
- Each value consists of two floats, the real part and the imaginary part
- Literal form (in Python):
- (7 + 3j), where 7 is the **real part** and 3 is the **imaginary part**

Primitive Data Types: Decimal

- For business applications (money)
  - Essential to COBOL
  - C# and F# also offers Decimal data type
- Decimal types are stored very much like character strings, using binary codes for the decimal(BCD)
- *Advantage*: accuracy
- *Disadvantages*: No exponents are allowed, range of values restricted, therefore sometimes wastes memory
- Examples,
  - C# has a built-in data type 'decimal', consisting of 128-bit resulting in 28-29 significant digits
  - Ruby's standard library includes a BigDecimal class in the module bigdecimal
  - Java's standard library includes a java.math.BigDecimal class

Primitive Data Types: Boolean

- Simplest of all
- Range of values: two elements, one for “true” and one for “false”
- Could be implemented as bits, but often as bytes
  - Advantage: readability

Primitive Data Types: Character

- Stored as numeric codings
- Most commonly used coding: ASCII
- An alternative, 16-bit coding: Unicode (UCS-2)
  - Includes characters from most natural languages
  - Originally used in Java
  - C# and JavaScript also support Unicode
- 32-bit Unicode (UCS-4)
  - Supported by Fortran, starting with 2003

Character String Data Types

- Values are sequences of characters
- Design issues:
  - Is it a primitive type or just a special kind of array?
  - Should the length of strings be static or dynamic?
- Typical operations:
  - Assignment and copying
  - Comparison ($=, >$, etc.)
  - Catenation
  - Substring reference
  - Pattern matching

Character String Variations

- **C and C++**
  - Not primitive
  - Use **char** arrays and a library of functions that provide operations
- **SNOBOL4** (a string manipulation language)
  - Primitive
  - Many operations, including **elaborate pattern matching**
- **Fortran and Python**
  - Primitive type with assignment and several operations
- **Java**
  - Primitive via the String class (`java.lang.string`)
- **Perl, JavaScript, Ruby, and PHP**
  - Provide built-in pattern matching, using regular expressions

Character String Length Variations

- Static: COBOL, Java's String class
- *Limited Dynamic Length*: C and C++
  - In these languages, a special character is used to indicate the end of a string's characters, rather than maintaining the length
- *Dynamic* (no maximum): SNOBOL4, Perl, JavaScript
- Ada supports all three string length options

Character String Implementation Design Choices

- **Static length:** compile-time descriptor
- Limited dynamic length: may need a run-time descriptor for length (but not in C and C++)
- **Dynamic length:** need run-time descriptor; allocation/deallocation is the biggest implementation problem

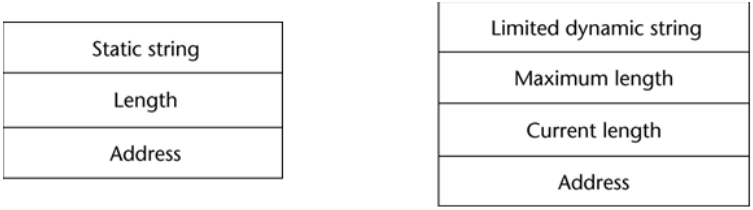

Figure 2: Compile-time vs. Run-time Descriptors for Strings

Enumeration Types

- All possible values, which are named constants, are provided in the definition
- C# example
  - **enum** $days(mon, tue, wed, thu, fri, sat, sun)$;
- **Design issues**
  - Is an 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?

Evaluation of Enumerated Type

- Aid to **readability**, e.g., no need to code a color as a number
- Aid to **reliability**, e.g., compiler can check:
  - operations (don't allow colors to be added)
  - No enumeration variable can be assigned a value outside its defined range
  - Ada, C#, and Java 5.0 provide better support for enumeration than C++ because enumeration type variables in these languages are not coerced into integer types

Array Types

- An array is a homogeneous **aggregation of data elements** in which an individual element is identified by its **position** in the aggregate, **relative** to the first element.
- Array Design Issues:
  - What types are legal for subscripts?
  - Are subscripting expressions in element references range checked?
  - When are subscript ranges bound?
  - When does allocation take place?
  - Are ragged or rectangular multidimensional arrays allowed, or both?
  - What is the maximum number of subscripts?
  - Can array objects be initialized?
  - Are any kind of slices supported?

Array Indexing

- *Indexing* (or subscripting) is a mapping from indices to elements
- array_name (index_value_list) an element
- Index Syntax
  - **Fortran and Ada** use parentheses
    - **Ada** explicitly uses parentheses to show uniformity between array references and function calls because both are *mappings*
  - Most other languages use brackets

Array Index (Subscript) Types

- **FORTRAN, C**: integer only
- **Ada**: integer or enumeration (includes Boolean and char)
- **Java**: integer types only
- Index range checking:
  - - C, C++, Perl, and Fortran do not specify range checking
  - - Java, ML, C# specify range checking
  - - In Ada, the default is to require range checking, but it can be turned off

Subscript Binding and Array Categories

- **Static**: subscript ranges are statically bound and storage allocation is static (before run-time)
  - **arrays with `static` modifier**
  - Advantage: efficiency (no dynamic allocation)
  - Example: In C and C++ static modifier are static
    ```
    static int myarray[3] = 2, 3, 4;
    ```
- **Fixed stack-dynamic**: subscript ranges are statically bound, but the allocation is done at declaration time
  - Example: **arrays without `static` modifier** are fixed stack-dynamic
    ```
    int array[3] = 2, 3, 4;
    ```
  - Advantage: space efficiency
- **Stack-dynamic**: subscript ranges are dynamically bound and the storage allocation is dynamic (done at run-time)

Subscript Binding and Array Categories *(Cont'd)*

- Advantage: flexibility (the size of an array need not be known until the array is to be used)
- Example: In **Ada**, you can use stack-dynamic arrays as -

```
Get(List_Len);
declare
List: array (1..List_Len) of Integer
begin
...
end;
```

- **Fixed heap-dynamic**: similar to fixed stack-dynamic: storage binding is **dynamic but fixed after allocation** (i.e., binding is done when requested and storage is allocated from heap, not stack)
  - Example: In **C/C++**, using `malloc/free` to allocate/deallocate memory from the heap
  - **Java** has fixed heap dynamic arrays
  - **C#** includes a second array class `ArrayList` that provides fixed heap-dynamic

Subscript Binding and Array Categories *(Cont'd)*

- **Heap-dynamic**: Binding of subscript ranges nd storage allocation is dynamic and can change any number of times
  - Advantage: flexibility (arrays can grow or shrink during program execution)
  - Examples: **Perl**, **JavaScript**, **Python**, and **Ruby** support heap-dynamic arrays
  - Perl: `@states = (\Idaho",\Washington",\Oregon");`
  - Python: `a = [1.25, 233, 3.141519, 0, -1]`

Heterogeneous Arrays

- A heterogeneous array is one in which the elements need not be of the same type
- Supported by Perl, Python, JavaScript, and Ruby
- Python example:
  - `a = array([12, 3.5, -1, 'two'])`

Heterogeneous Arrays

- C-based languages
  - `int list [] = 1, 3, 5, 7`
  - `char *names [] = \Mike", \Fred",\Mary Lou";`
- Ada
  - `List :  array (1..5) of Integer :=`
    `(1 => 17, 3 => 34, others => 0);`
- Python
  - List comprehensions
  - `list = [x ** 2 for x in range(12) if x % 3 == 0]`
    `puts [0, 9, 36, 81] in list`

Thanks

Thanks for your time and attention!

kamruddin.nur@northsouth.edu
researchgate.net/profile/Kamruddin_Nur