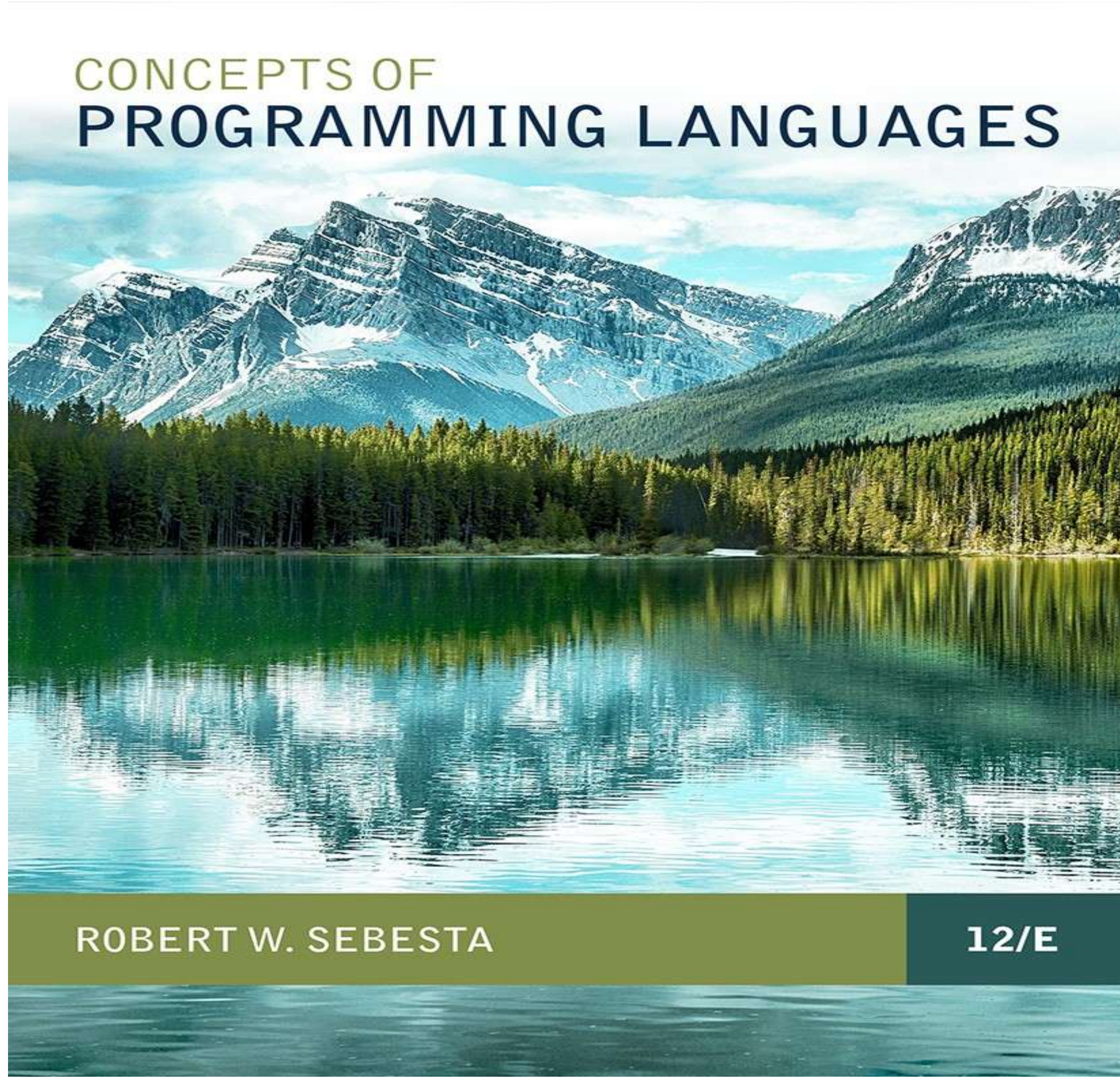
Lecture 3

Chapter 6

Data Types

Part 1

First Semester
1447 - 2025



ISBN 0-321—49362-1

Lecture 3 Topics:

- Introduction
- Primitive Data Types
- Character String Types
- Enumeration Types
- Array Types
- Associative Arrays
- Record Types
- Tuple Types
- List Types
- Union Types
- Pointer and Reference Types
- Optional Types
- Type Checking
- Strong Typing
- Type Equivalence
- Theory and Data Types

Introduction

- A **data type** defines:
  - A collection of **data** objects.
  - A set of predefined **operations** on those objects.
- A **descriptor** is the collection of the attributes of a variable.
- An **object** represents an instance of a user-defined (abstract data) type.
- One design issue for **all data types**:
  - What **operations** are defined and how are they specified?

Primitive Data Types

- 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 <u>eight</u> different integer types in a language.
- Java's <u>signed</u> integer sizes:
  - **byte**
  - **short**
  - **int**
  - **long**

Primitive Data Types: Floating Point

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

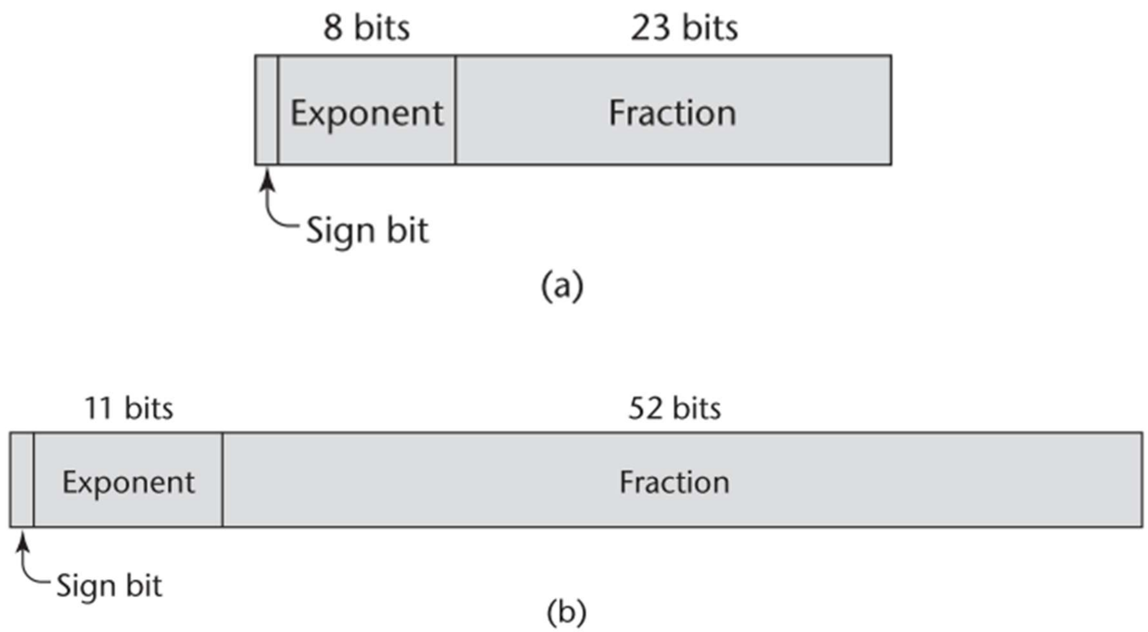


Figure: IEEE floating-point formats: **(a)** single precision, **(b)** double precision

Primitive Data Types: Complex

- Some languages support a **complex type**, e.g.,
  - C99
  - Fortran
  - Python
- Each value consists of <u>two floats</u>:
  - The *real* part
  - 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# offers a decimal data type.
- Store a fixed number of decimal digits, in coded form called **binary coded decimal** (BCD).
- **Advantage**:
  - accuracy
- **Disadvantages**:
  - limited range
  - wastes memory

Primitive Data Types: Boolean

- Simplest of all.
- Range of values:
  - Two elements, one for “**true**” and one for “**false**”.
- Could be implemented as <u>bits</u>, but often as <u>bytes</u>.
  - **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.
  - Now supported by many languages.
- **32-bit** Unicode (**UCS-4**)
  - Supported by Fortran, starting with 2003.

Character String 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?

Character String Types Operations

- Typical **operations**:
  - Assignment and copying.
  - Comparison (=, >, etc.).
  - Catenation.
  - Substring reference (aka slicing).
  - Pattern matching.

Character String Type in Certain Languages

- **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** (and **C#**, **Ruby**, and **Swift**):
  - Primitive via the String class.
- **Perl**, **JavaScript**, **Ruby**, and **PHP**:
  - Provide built-in pattern matching, using regular expressions.

Character String Length Options

- **Static Length**:
  - 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) Length**:
  - SNOBOL4, Perl, JavaScript.

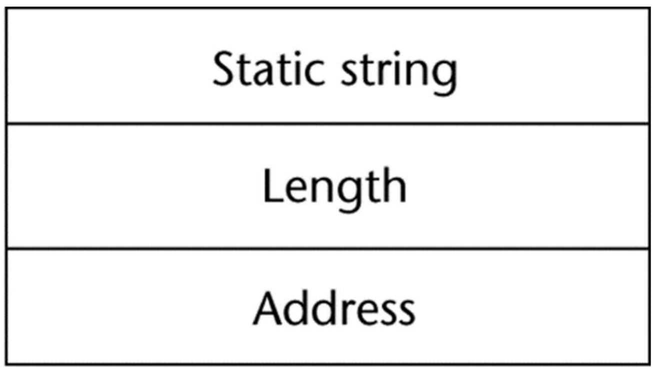
Character String Type Evaluation

- Aid to writability.
- As a **primitive type** with **static length**, they are <u>inexpensive</u> to provide.
  - Why not have them?
- **Dynamic length** is nice, but is it worth the expense?

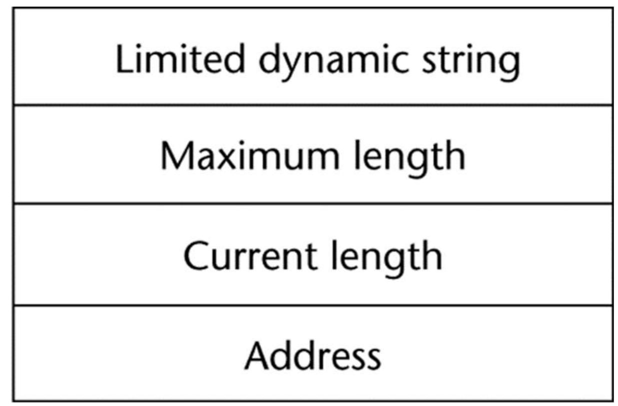
Character String Implementation

- **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.

Compile-Time & Run-Time Descriptors



Compile-time descriptor for static strings.



Run-time descriptor for limited dynamic strings.

User-Defined Ordinal Types

- An **ordinal type** is one in which the range of possible values can be easily associated with the set of positive integers.
- **Examples** of **primitive ordinal types** in **Java**:
  - integer
  - char
  - boolean

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.
  - **C#**, **F#**, **Swift**, 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 aggregate of data elements in which an individual element is identified by its position in the aggregate, relative to the first element.

- In many languages, such as **C**, **C++**, **Java**, and **C#**, all the elements of an array are required to be of the same type.

- **C#** and **Java 5.0** provide generic arrays, that is, arrays whose elements are references to objects, through their class libraries.

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 `[]`.

Arrays Index (Subscript) Types

- **FORTRAN**, **C**:
  - integer only.
- **Java**:
  - integer types only.
- **Index range checking**:
  - **C**, **C++**, **Perl**, and **Fortran** <u>do not</u> specify range checking.
  - **Java**, **ML**, **C#** specify range checking.

Subscript Binding & Array Categories

(1) Static array:

- Subscript ranges are statically bound.
- Storage allocation is static (before run-time).
- **Advantage**:
  - Efficiency (no dynamic allocation).

(2) Fixed stack-dynamic array:

- Subscript ranges are statically bound.
- Storage allocation is done at declaration time.
- **Advantage**:
  - Space efficiency.

Subscript Binding & Array Categories (cont.)

(3) Fixed heap-dynamic array:

- 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).

Subscript Binding & Array Categories (cont.)

- **(4) Heap-dynamic array**:
  - Binding of subscript ranges and storage allocation is dynamic and can change any number of times.
  - **Advantage**:
    - Flexibility (arrays can grow or shrink during program execution).

Subscript Binding & Array Categories (cont.)

- **C** and **C++ arrays** that include static modifier are **static**.
- **C** and **C++ arrays** without static modifier are **fixed stack-dynamic**.
- **C** and **C++** provide **fixed heap-dynamic** arrays.
- **C#** includes a second array class `ArrayList` that provides **fixed heap-dynamic**.
- **Perl**, **JavaScript**, **Python**, and **Ruby** support **heap-dynamic arrays**.

Array Initialization

- Some language allow initialization at the time of storage allocation:
  - **C**, **C++**, **Java**, **Swift**, and **C#**:
    - **C#** example:
      - `int list[] = {4, 5, 7, 83}`
    - Character strings in **C** and **C++**:
      - `char name[] = "freddie";`
    - Arrays of strings in **C** and **C++**:
      - `char *names[] = {"Bob", "Jake", "Joe"];`
    - **Java** initialization of String objects:
      - `String[] names = {"Bob", "Jake", "Joe"};`

Array Initialization

- **C-based languages**:
  - `int list [] = {1, 3, 5, 7};`
  - `char *names [] = {"Mike", "Fred", "Mary Lou"};`
- **Python**:
  - List comprehensions:
    - `list = [x ** 2 for x in range(12) if x % 3 == 0]`
    - `puts [0, 9, 36, 81] in list`

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**.

Any Questions?

- Please, read the relevant sections in **chapter 6**.
- I hope you were taking some **notes**!
- To test your understanding of this lecture, have a go with the "**Review Questions**" in page 294 of the textbook.
- Please, keep **reviewing** this lecture regularly.
- I hope you're doing well with the assignment!