Array Types

* An array is a homogeneous aggregate where a value is defined by its position

Array Initialization

* Some languages allow initialization at the time of storage allocation

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
* mylist = [6, “egg”, 5.3]

Rectangular and Jagged Arrays

* A **rectangular array** is a multi-dimensioned array in which all of the rows have the same number of elements and all columns have the same number of elements
* A **jagged matrix** has rows with varying numbers of elements
  + Possible when multi-dimensioned arrays actually appear as arrays of arrays
* C, C++, and Java support jagged arrays
* F# and C# support both rectangular arrays and jagged arrays

Slices

* A **slice** is some substructure of an array; nothing more than a referencing mechanism

Implementation of Arrays

* Access function maps subscript expressions to an address in the array

Accessing Multi-Dimensioned Arrays

* Two common ways:
  + Row major order (by rows) – used in most languages
  + Column major order (by column)

Associative Arrays

* An **associative array** is an unordered collection of data elements that are indexed by an equal number of values called *keys*
  + User-defined keys must be stored
* Design issues:
  + What is the form of references to elements?
  + Is the size static or dynamic?
* Built in type in Perl, Python, Ruby, and Lua
  + In Lua, they are supported by tables

Record Types

* A **record** is a possibly **heterogeneous** aggregate of data elements in which the individual elements are identified by names
* In some languages that support object-oriented programming, data classes serve as records

Pointer and Reference Types

* A **pointer** type variable has a range of values that consists of memory addresses and a special value, *null*
* Provide the power of indirect addressing
* Provide a way to manage dynamic memory
* A pointer can be used to access a location in the area where storage is dynamically created (usually called a heap)

Pointer Operations

* Two fundamental operations: assignment and dereferencing
* Assignment is used to set a pointer variable’s value to some useful address

Problems with Pointers

* Dangling Pointers (dangerous)
  + A pointer points to a heap-dynamic variable that has been deallocated
* Lost heap-dynamic variable
  + An allocated heap-dynamic variable that is no longer accessible to the user program (often called garbage)
    - Pointer p1 is set to point to a newly created heap-dynamic variable
    - Pointer p1 is later set to point to another newly created heap-dynamic variable
    - The process of losing heap-dynamic variables is called *memory leakage*