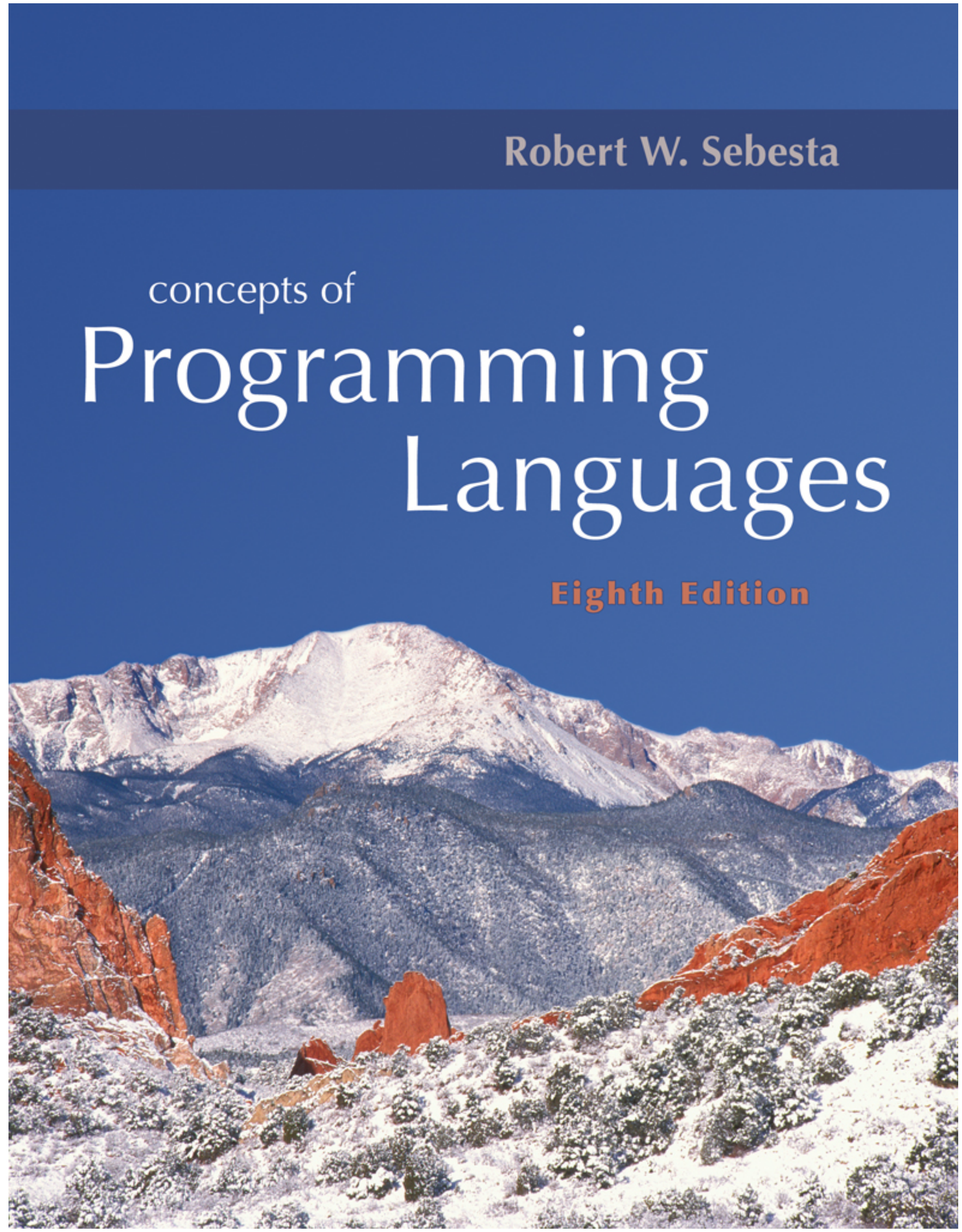
Chapter 6
part 1

Data Types

(updated based on 11th edition)



ISBN 0-321-49362-1

Chapter 6 Topics

- Introduction
- Primitive Data Types
- Character String Types
- User-Defined Ordinal Types
- Array Types
- Associative Arrays
- Record Types
- Union Types
- Pointer and Reference Types

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

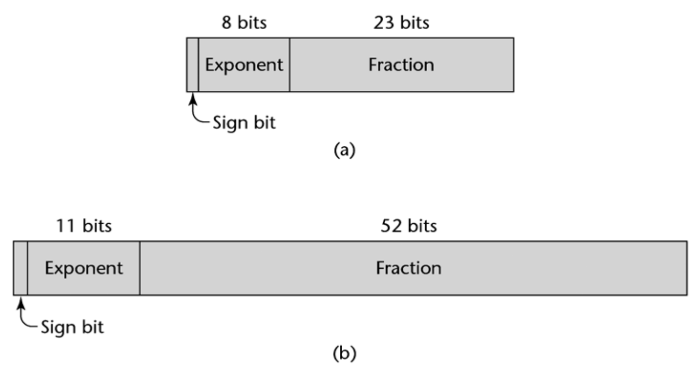
- Almost always an exact reflection of the hardware so the mapping is trivial
- There may be as many as eight different integer types in a language
- Java's signed integer sizes: byte, short, int, long
- Example integer type of unlimited length not supported by hardware?

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

Figure 6.1

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



Primitive Data Types: Complex

- Some languages support a complex type, e.g., 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# offers a decimal data type
- Store a fixed number of decimal digits, in coded form (BCD) as character strings
- 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 bits, but often as bytes
  - Advantage: readability

Primitive Data Types: Character

- Stored as numeric codings
- Most commonly used 8-bit coding: ASCII
- An alternative, 16-bit coding: Unicode
  - Includes characters from most natural languages
  - Originally used in Java
  - C#, JavaScript, Python, Perl also support Unicode

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
  - 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
  - Primitive via the String class
- Perl, JavaScript, Ruby, and PHP
- – Provide built-in pattern matching, using regular
- expressions

Character String Length Options

- Static: COBOL, Java's String class, Python
- Limited Dynamic Length (fixed maximum): 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

We showed Python and Ruby string examples

Character String Type Evaluation

- Aid to 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?

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/de-allocation is the biggest implementation problem

Compile- and Run-Time Descriptors

Static string
Length
Address

Compile-time descriptor for static strings

Limited dynamic string
Maximum length
Current length
Address

Run-time descriptor for limited dynamic 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
  - Book says not supported in recent scripting languages such as Python, but has been added to standard Python library in 2013

Subrange Types

- An ordered contiguous subsequence of an ordinal type
  - Example: 12..18 is a subrange of integer type
- Ada's design
  - type Days is (mon, tue, wed, thu, fri, sat, sun);
  - subtype Weekdays is Days range mon..fri;
  - subtype Index is Integer range 1..100;

  - Day1: Days;
  - Day2: Weekday;
  - Day2 := Day1;

Subrange Evaluation

- Aid to readability
  - Make it clear to the readers that variables of subrange can store only certain range of values
- Reliability
  - Assigning a value to a subrange variable that is outside the specified range is detected as an error

Implementation of User-Defined Ordinal Types

- Enumeration types are implemented as integers
- Subrange types are implemented like the parent types with code inserted (by the compiler) to restrict assignments to subrange variables

Array Types

- An array is an aggregate of homogeneous 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?
- 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, PL/I, 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
- 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)
  - Advantage: efficiency (no dynamic allocation)
- Fixed stack-dynamic: subscript ranges are statically bound, but the allocation is done at declaration time
  - Advantage: space efficiency

Subscript Binding and Array Categories (cont.)

- Fixed Heap-dynamic: subscript ranges are dynamically bound and the storage allocation is dynamic (done at run-time from heap). But both are still fixed after storage is allocated.
  - Advantage: flexibility (the size of an array need not be known until the array is to be used)

Subscript Binding and Array Categories (cont.)

- Heap-dynamic: 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 and 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

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

Array Initialization

- Some language allow initialization at the time of storage allocation
  - C, C++, Java, 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”};

Arrays Operations

- APL provides the most powerful array processing operations for vectors and matrixes as well as unary operators (for example, to reverse column elements)
- Ada allows array assignment but also catenation
- Python's array assignments, but they are only reference changes. Python also supports array catenation and element membership operations
- Ruby also provides array catenation
- Fortran provides elemental operations because they are between pairs of array elements
  - For example, + operator between two arrays results in an array of the sums of the element pairs of the two arrays

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 number of elements
  - Possible when multi-dimensioned arrays actually appear as arrays of arrays
- C, C++, and Java support jagged arrays
- Fortran, Ada, and C# support rectangular arrays (C# also supports jagged arrays)

Slices

- A slice is some substructure of an array; nothing more than a referencing mechanism
- Slices are only useful in languages that have array operations

Slice Examples

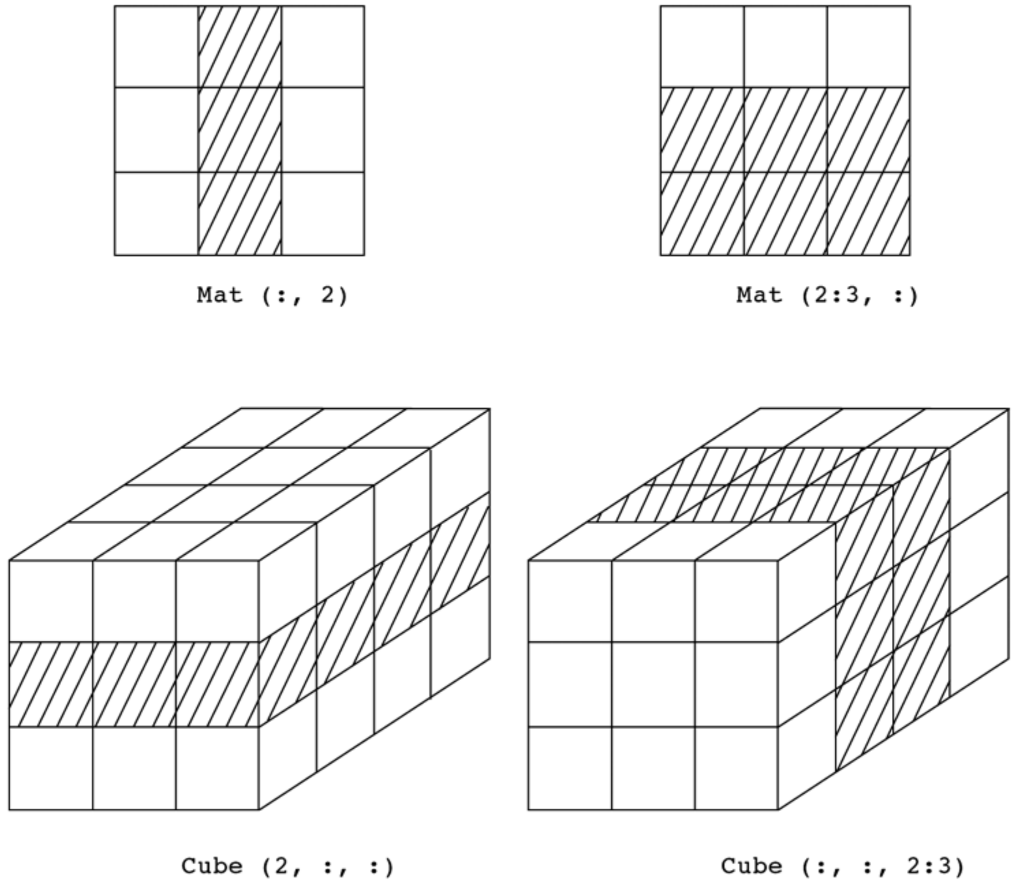
- Fortran 95
  - Integer, Dimension (10) :: Vector
  - Integer, Dimension (3, 3) :: Mat
  - Integer, Dimension (3, 3) :: Cube

  - Vector (3:6) is a four element array

Slices Examples in Fortran 95

Figure 6.4

Example slices in Fortran 95



Similar in Matlab; Python's numpy

Implementation of Arrays

- Access function maps subscript expressions to an address in the array
- Access function for single-dimensioned arrays:

- **address(list[k]) = address (list[lower_bound]) +
  ((k-lower_bound) * element_size)**

Implementation of Arrays

Figure 6.5

Compile-time description for single-dimensioned arrays

Array
Element type
Index type
Index lower bound
Index upper bound
Address

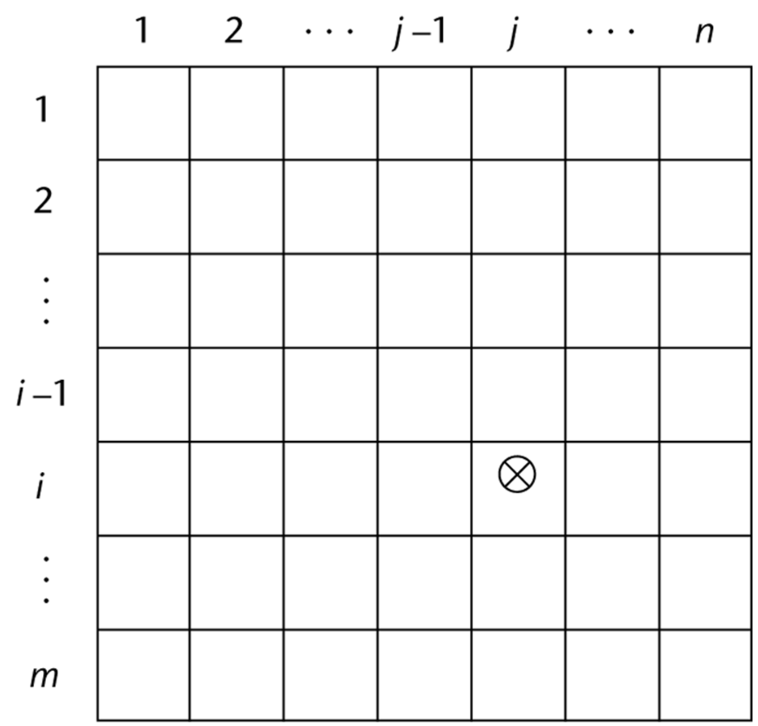
Accessing Multi-dimensioned Arrays

- Hardware memory linear, so values of data types with two or more dimensions (eg, matrix) must be mapped onto single dimension array)

- Two common ways:
  - Row major order (by rows) – used in most languages
  - column major order (by columns) – used in Fortran

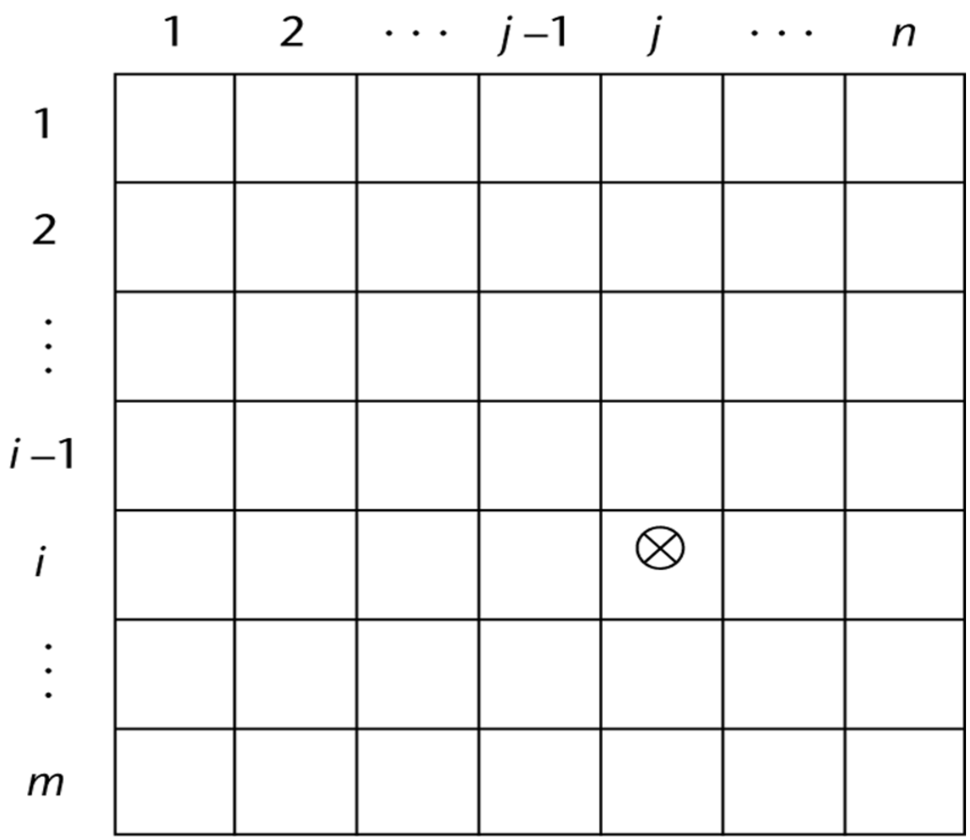
Locating an Element in a Multi-dimensioned Array

location (a[i,j]) = address of a [1,1] +
((((number of rows above the ith row) *
(size of a row)) +
(number of elements left of the jth column)) *
element_size)



Locating an Element in a Multi-dimensioned Array (2)

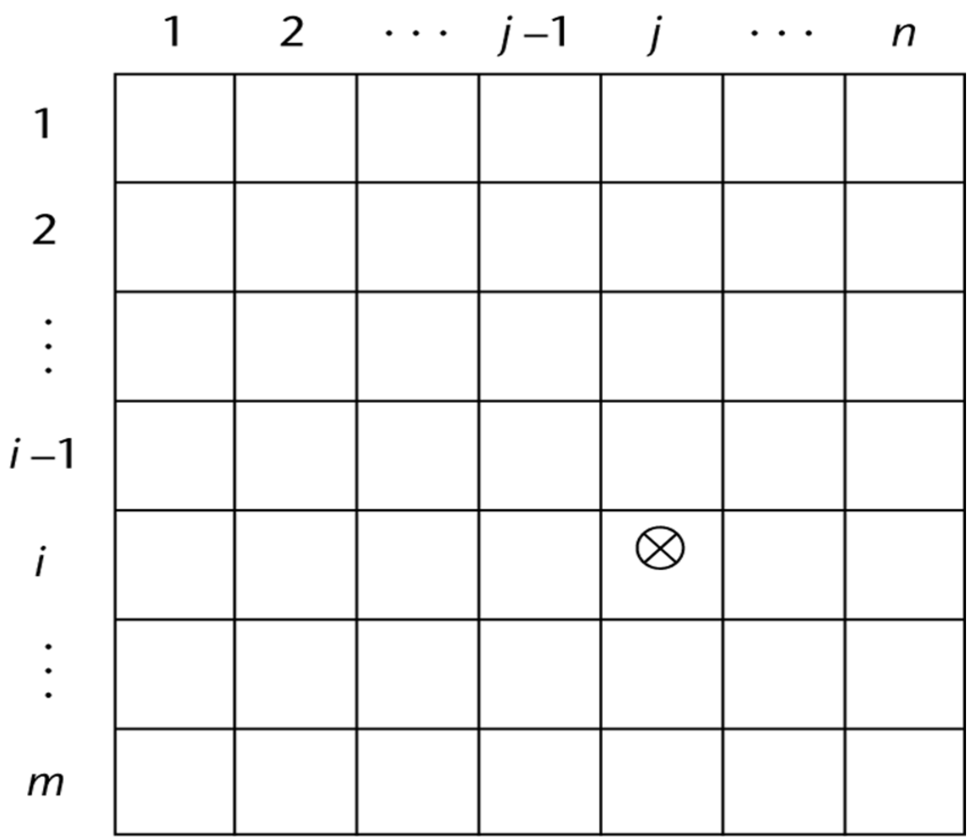
location (a[i,j]) = address of a [1,1] +
((((i-1) * n) + (j-1)) * element_size)



Locating an Element in a Multi-dimensioned Array (3)

location (a[i,j]) = address of a [1,1] −((n+1) * element_size) + (i*n+j)) * element_size)

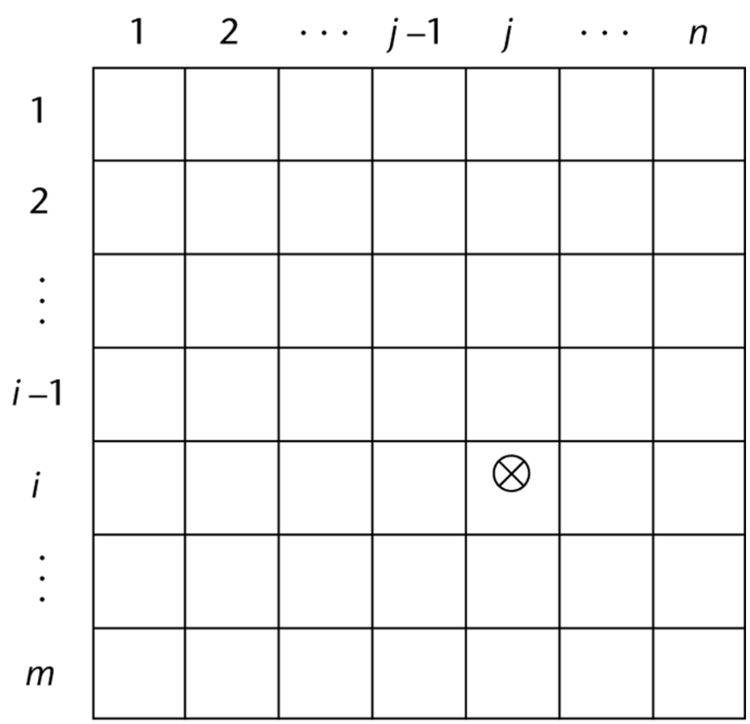
Rearranged so first two terms constant



Locating an Element in a Multi-dimensioned Array

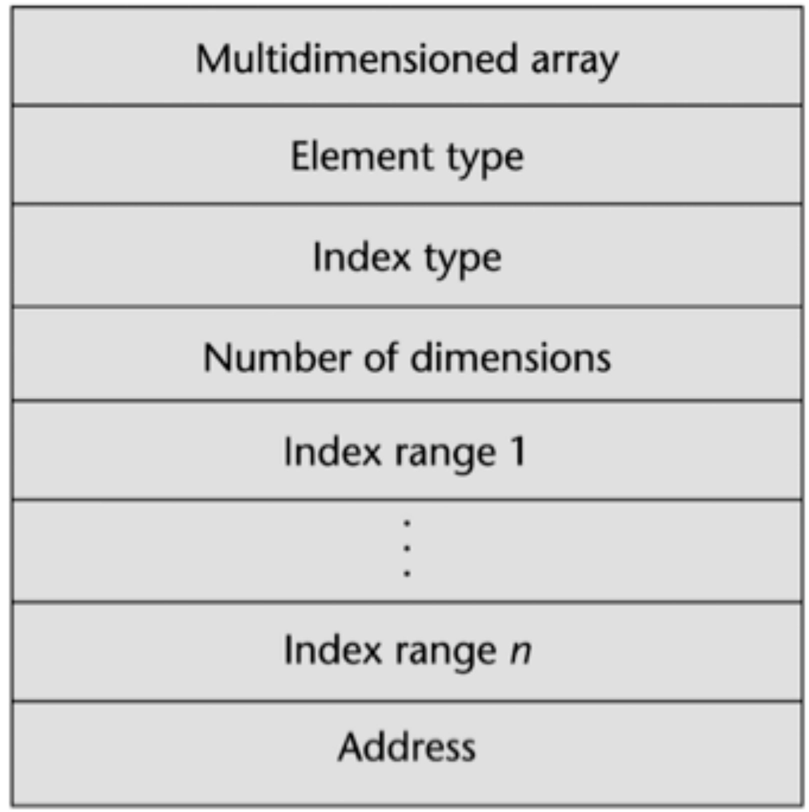
- General format

Location (a[i,j]) = address of a [row_lb,col_lb] + (((i – row_lb) * n) + (j – col_lb)) * element_size



Compile-Time Descriptors

Single-dimensioned array



Multi-dimensional array

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?

Associative Arrays in Perl

- Names (hash variable) begin with **%;** literals are delimited by parentheses
  - %hi_temps = ("Mon" => 77, "Tue" => 79, “Wed” => 65, …);
- Subscripting is done using braces and keys
  - $hi_temps{"Wed"} = 83;
  - Elements can be removed with delete
  - delete $hi_temps{"Tue"};

We showed Ruby example