What is a Data Structure?

- Data is a collection of facts, such as values, numbers, words, measurements, or observations.
- Structure means a set of rules that holds the data together.
- A data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently.
  - Different kinds of data structures are suited to different kinds of applications, and some are highly specialized to specific tasks.
  - Data structures provide a means to manage huge amount of data efficiently.
  - Usually, efficient data structures are a key to designing efficient algorithms.
  - Data structures can be nested.

Types of Data Structures

- Data structures are classified as either
  - Linear (*e.g*, arrays, linked lists), or
  - Nonlinear (*e.g*, trees, graphs, etc.)
- A data structure is said to be linear if it satisfies the following four conditions
  - There is a unique element called the first
  - There is a unique element called the last
  - Every element, except the last, has a unique successor
  - Every element, except the first, has a unique predecessor
- There are two ways of representing a linear data structure in memory
  - By means of sequential memory locations (arrays)
  - By means of pointers or links (linked lists)

Arrays

- Data is often available in tabular form
- Tabular data is often represented in arrays
- Matrix is an example of tabular data and is often represented as a 2-dimensional array
  - Matrices are normally indexed beginning at 1 rather than 0

Properties of Arrays

- The components of an array are all of the same type.
- Array is a random access data structure.
- Array is a static data structure.
- Access time for an array element is constant, that is, $O(1)$.
- An array is a suitable structure when a small number of insertions and deletions are required.
- An array is a suitable structure when a lot of searching and retrieval are required.

Parameters of Arrays

- **Base Address (b):** The memory address of the first byte of the first array component.
- **Component Length (L):** The memory required to store one component of an array.
- **Upper and Lower Bounds (l_i, u_i):** Each index type has a smallest value and a largest value.
- **Dimension**

Representation of Arrays in Memory

- **Array Mapping Function (AMF)**
  - AMF converts index value to component address

- **Linear (1 D) Arrays:**

  a : array [l_1 .. u_1] of element_type

  $$\text{Then addr}(a[i]) = b + (i - l_1) \times L$$
  $$= c_0 + c_1 \times i$$

  Therefore, the time for calculating the address of an element is same for any value of i.

Representation of Arrays in Memory

- **Array Mapping Function (AMF)**
  - AMF converts index value to component address

- **2 D Arrays:**

  a : array $[l_1 \,..\, u_1, l_2 \,..\, u_2]$ of element_type

  In which order are the elements stored?
  - Row major order (C, C++, Java support it)
  - Column major order (Fortran supports it)

Representation of Arrays in Memory

Row Major Order:

a[0][0]	a[0][1]	a[0][2]	a[0][3]	row 0
a[1][0]	a[1][1]	a[1][2]	a[1][3]	row 1
a[2][0]	a[2][1]	a[2][2]	a[2][3]	row 2

Representation of Arrays in Memory

- **2 D Arrays:**

The elements of a 2-dimensional array a are declared as:

int a[3][4];

may be shown as a table

a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]

Representation of Arrays in Memory

- **Array Mapping Function (AMF)**
  - AMF converts Index value to component address

- **2 D Arrays (Column Major Order):**

  a : array $[l_1 \,..\, u_1, l_2 \,..\, u_2]$ of element_type

$$\text{Then addr}(a[i, j]) = b + (j - l_2)\times(u_1 - l_1 + 1)\times L + (i - l_1)\times L$$
$$= c_0 + c_1\times i + c_2\times j$$

Therefore, the time for calculating the address of an element is same for any value of (i, j).

Representation of Arrays in Memory

- **Array Mapping Function (AMF)**
  - AMF converts Index value to component address

- **2 D Arrays (Row Major Order):**

  a : array $[l_1 \,..\, u_1, l_2 \,..\, u_2]$ of element_type

$$\text{Then addr}(a[i, j]) = b + (i - l_1)\times(u_2 - l_2 + 1)\times L + (j - l_2)\times L$$
$$= c_0 + c_1\times i + c_2\times j$$

Therefore, the time for calculating the address of an element is same for any value of (i, j).

Representation of Arrays in Memory

- **Array Mapping Function (AMF)**
  - AMF converts Index value to component address

- **3 D Arrays :**

  a : array $[l_1 \,..\, u_1, l_2 \,..\, u_2, l_3 \,..\, u_3]$ of element_type

$$\text{Then addr}(a[i, j, k]) = b + (i - l_1)\times(u_2 - l_2 + 1)\times(u_3 - l_3 + 1)\times L + (j - l_2)\times(u_3 - l_3 + 1)\times L + (k - l_3)\times L$$
$$= c_0 + c_1\times i + c_2\times j + c_3\times k$$

Therefore, the time for calculating the address of an element is same for any value of (i, j, k).