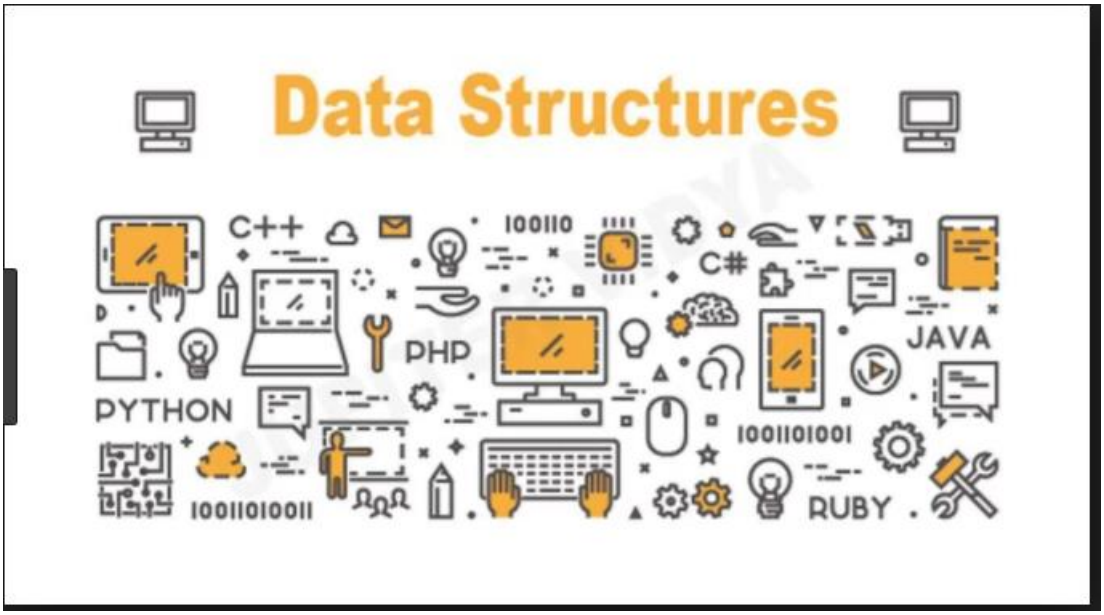
College of Computer Science and information Technology

Data Structures



Lecturer: Dr. Raidah Salim

Course objectives

This course aims to make the student capable of understanding and writing different data structures as:

- Array
- String
- Linked List
- Stack
- Queue

Introduction

In computer science, a data structure is a particular way of organizing and storing data in a computer so that it can be accessed and modified efficiently.

More precisely, a data structure is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data.

Basic types of Data Structures

Anything that can store data can be called as a data structure, hence integer, float, boolean, char etc, all are data structures. They are known as **Primitive Data Structures**.

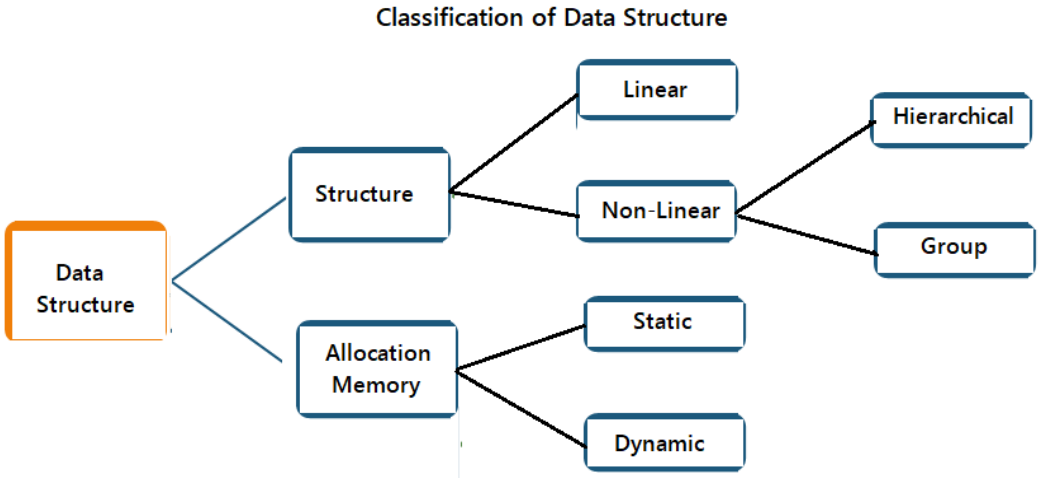
Then we also have some **complex Data Structures**(**Abstract Data Structures)**, which are used to store large and connected data. Some example of Abstract Data Structures are :

- Linked List
- Tree
- Graph
- Stack, Queue

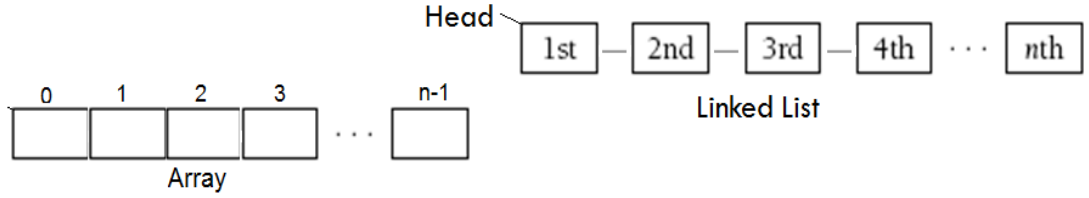
The data structures can also be classified on the basis of the following characteristics:

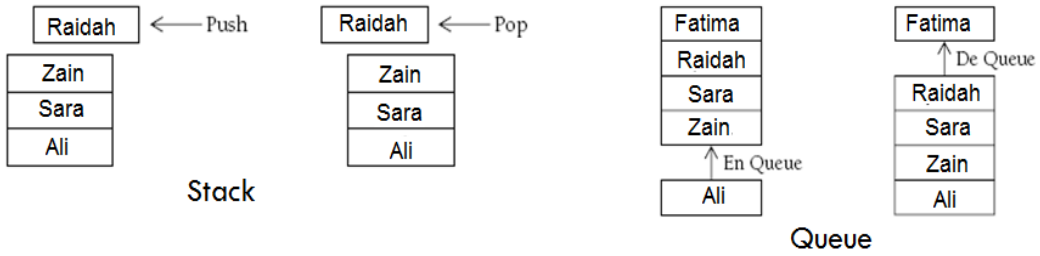
Characteristic	**Description**
Linear	In Linear data structures, the data items are arranged in a linear sequence. Example: **Array**
Non-Linear	In Non-Linear data structures, the data items are not in sequence. Example: **Tree**, **Graph**
Homogeneous	In homogeneous data structures, all the elements are of same type. Example: **Array**
Non-Homogeneous	In Non-Homogeneous data structure, the elements may or may not be of the same type. Example: **Structures**

Static	Static data structures are those whose sizes and structures associated memory locations are fixed, at compile time. Example: **Array**
Dynamic	Dynamic structures are those which expands or shrinks depending upon the program need and its execution. Example: **Linked List**



Classification according to *structure* :In these data structures the elements form a sequence, or the elements do not form a sequence.





■

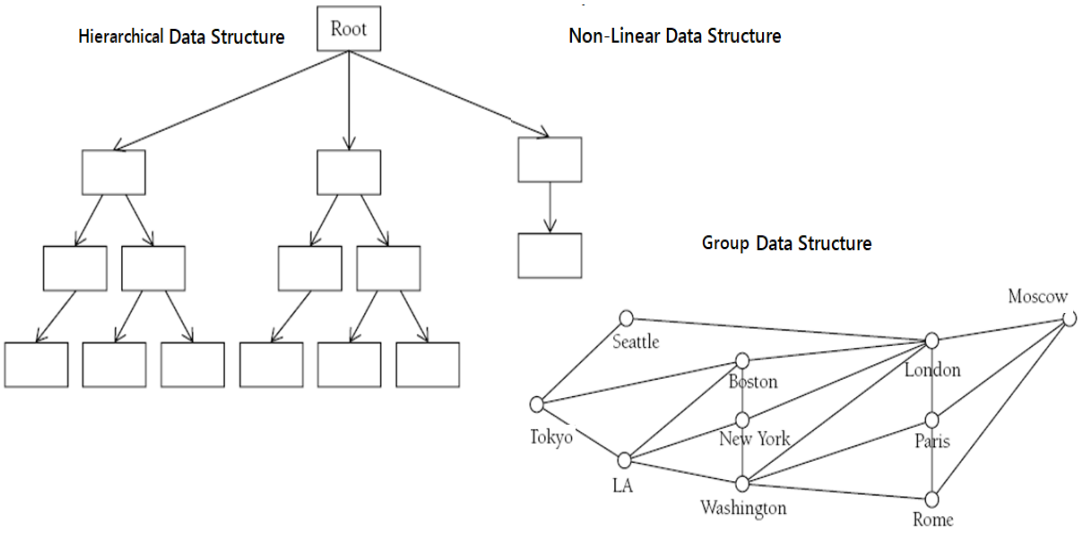
Classification according to *Allocation memory* :

Static memory allocation means the program must obtain its space before the execution and cannot obtain more while or after execution.

Example: Arrays

Dynamic memory allocation is the ability for a program to obtain more memory space at execution time to hold new nodes and to release space no longer needed.

Example: Linked Lists, Stacks, Queues and Trees



All these data structures allow us to perform different operations on data:

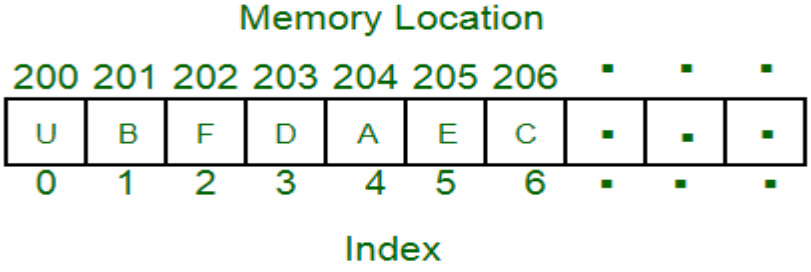
1. **Traversing:** It is used to access each data item exactly once.
2. **Searching**: It is used to find out the location of the data item.
3. **Inserting**: It is used to add a new data item in the given collection of data items.
4. **Deleting:** It is used to delete an existing data item from the given collection of data items.
5. **Sorting**: It is used to arrange the data items in some order i.e. in ascending or descending order.

The data structures can be viewed in two ways, **physically** and **logically**.

- The **physical** data structure refers to the physical arrangement of the data on the memory.
- The **logical** data structure concerns how the data "seem" to be arranged and the meanings of the data elements in relation to one another.

Arrays data structure

An array is collection of items stored at continuous memory locations. The idea is to store multiple items of same type together. This makes it easier to calculate the position of each element by simply adding an offset to a base value(the memory location of the first element of the array). Each element can be uniquely identified by their index in the array.



Types of indexing in array:

- 0 (zero-based indexing): The first element of the array is indexed by subscript of 0
- 1 (one-based indexing): The first element of the array is indexed by subscript of 1
- n (n-based indexing): The base index of an array can be freely chosen. Usually programming languages allowing n-based indexing also allow negative index values and other scalar data types like enumerations, or characters may be used as an array index.

Features of Arrays

An array is :

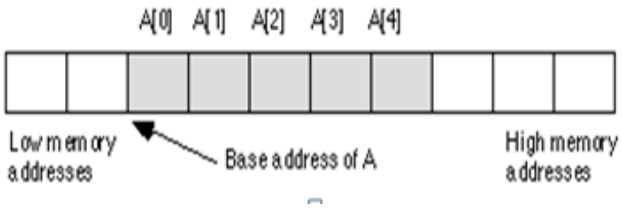
- **Linear** data structure: means organize in memory as linear order.
- **homogeneous** structure: all components in the structure are of the same data type.
- **finite** structure **:** indicates that there is a **last element**.
- **fixed size** structure**:** mean that the size of the array must be known at **compile time**.
- **contiguously** data structure: means that there is a first element, a second element, and so on.

- The **component selection mechanism** of an array is ***direct access(random access),*** which means we can access any element directly (by using specific equation), without first accessing the preceding elements. This makes accessing elements by position faster.

A one-dimensional array is homogeneous structure, it can be visualized as a list. **Logical structure** for one-dimensional array with 15 elements as shown below:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	7	9	12	15	23	30	45	70	77	80	88	90	99	

Physical structure for one-dimensional array with five elements will appear in memory as shown below:



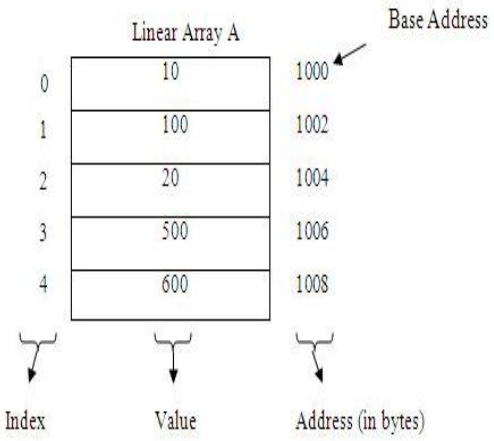
We can find out the *location of any element* by using following formula:

Loc (ArrayName [k]) = Loc (ArrayName [1]) + (k-LB)* w

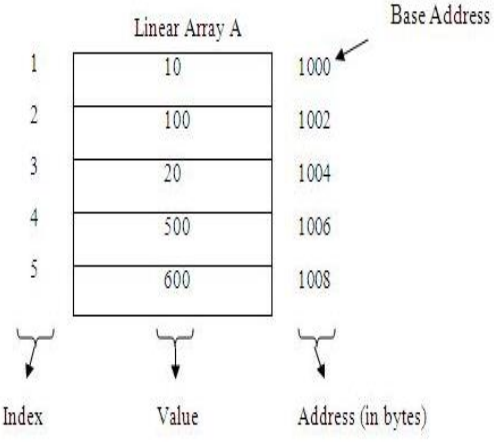
where:
Loc (ArrayName [k]): is the address of the kth element of ArrayName.
Loc (ArrayName [1]): is the base address or address of first element of ArrayName.
w : is the number of bytes taken by one element(element size).
LB : is the lower bound.

Example1 : Suppose we want to find out Loc (A [3]) that store as the following figure in two case (LB=1 and LB=0), for it, we have: Base(A)=1000, w = 2 bytes.

LB=0



LB=1

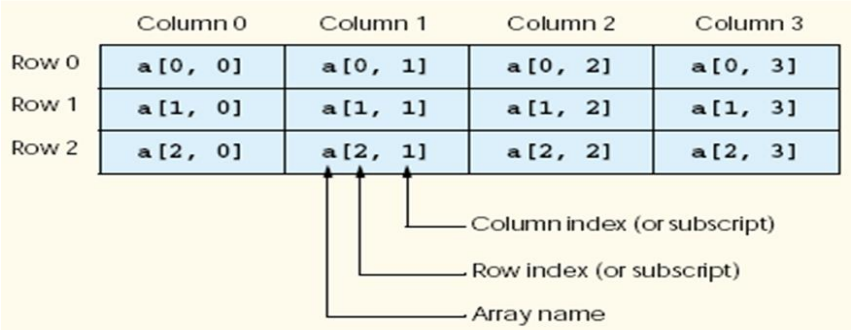


in Case LB = 1
LOC(A[3])=1000 + 2 (3 – 1)
= 1000 + 2 (2)
= 1000 + 4
= 1004

If Case LB = 0
LOC(A[3])=1000 + 2 *3
= 1000 + 6
= 1006

H.w. : Suppose the following declaration:
int A[6];
1. Draw the logical structure and physical structure for the array A.
2. Find Loc(A(5)) , where the first element of the array A =200.

A two-dimensional array is also homogeneous structure, it can be visualized as a table consisting of rows and columns. The element in a two-dimensional array is accessed by specifying the row and column indexes of the item in the array. **Logical Level** for two-dimensional array with 3(0-2) rows and 4(0-3)column elements will appear as shown below:

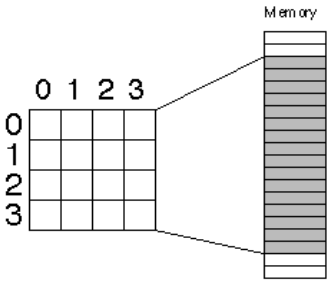


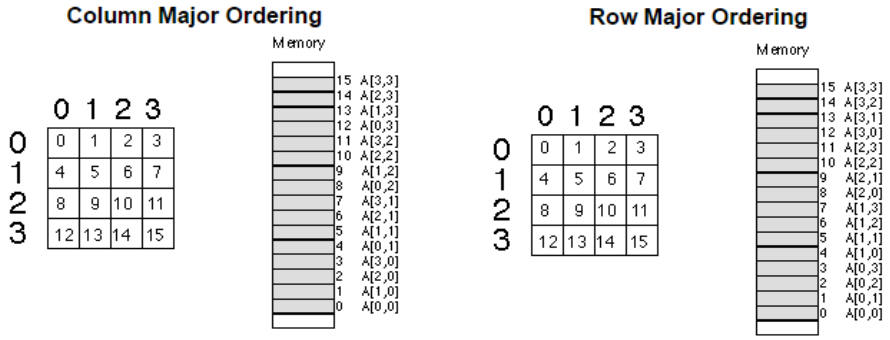
In the computer memory, all elements are stored linearly using contiguous addresses. Therefore, in order to store a two-dimensional matrix, two dimensional address space must be mapped to one-dimensional address space. There two methods for arranging two or multidimensional :

1. Row-major order

2. Column-major order

For example the physical structure for array with 3X3 elements will appear in memory as shown below in two methods:





In row-major order:

- consecutive elements of the **rows** of the array are contiguous in memory.
- Used in C, C++, PL/I, Pascal, Python and Java.

- in **column-major order**:
  - consecutive elements of the columns are contiguous.
  - Used in Fortran, OpenGL and Matlab.

Example: The following array:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$$

A would be stored as follows in the two orders:

Column-major order
e.g., Fortran

Address	Value
0	a_{11}
1	a_{21}
2	a_{12}
3	a_{22}
4	a_{13}
5	a_{23}

Row-major order
e.g., C++

Address	Value
0	a_{11}
1	a_{12}
2	a_{13}
3	a_{21}
4	a_{22}
5	a_{23}

We can find out the location of any element in array (NXM) by using following formulas:

1. **In case of Row Major Order:**

Loc (A [i, j]) = Loc (A [1,1]) +([i-1]*M+ [j-1]) * w

where:

Loc(A[[i,j]) : is the location of the element in the ith row and jth column.

Loc (A [1,1]) : is the base address or address of the first element of the array A.

w : is the number of bytes required to store single element of the array A.

M : is the total number of columns in the array.

Example: finding the location (address) of element in 2D

Suppose A $_{3 \times 4}$ (N=3 and M=4) integer array A is show as below and base address = 1000 and number of bytes=2. find the location of A [3,2]:

A:

10	20	50	60
90	40	30	80
75	55	65	79

Address	Elements
1000	10
1002	20
1004	50
1006	60
1008	90
1010	40
1012	30
1014	80
1016	75
1018	55
1020	65
1022	79

LOC (A [3,2]) = 1000 + 2 [4 (3-1) + (2-1)]
= 1000 + 2 [4 (2) + 1]
= 1000 + 2 [8 + 1]
= 1000 + 2 [9]
= 1000 + 18
= 1018

2. **In case of Column Major Order**:

Loc (A [i,j]) = Loc (A [1, 1]) + ([j-1]*N+ [i-1])*w

where

Loc(A[i,j]): is the location of the element in the ith row and jth column.

Loc (A [1,1]): is the base address or address of the first element of the array A.

w: is the number of bytes required to store single element of the array A.

N: is the total number of rows in the array.

Example: finding the location (address) of element in 2D

Suppose A $_{3 \times 4}$ (N=3 and M=4) integer array A and base address =1000 and number of bytes=2. find the location of A [3,2]:

Address	Elements
1000	10
1002	90
1004	75
1006	20
1008	40
1010	55
1012	50
1014	30
1016	65
1018	60
1020	80
1022	79

LOC (A [3,2]) = 1000 + 2 [3 (2-1) + (3-1)]
= 1000 + 2 [3 (1) + 2]
= 1000 + 2 [3 + 2]
= 1000 + 2 [5]
= 1000 + 10
= 1010

Note: if the value of w not determine, it suppose equal to 1.

H.W.

1. You have the matrix A [3, 4] and the base address is 1500. By using rows major order:
a. Draw logical structure and physical structure of the matrix A.
b. find the address of the element A [2, 3].

2. You have the matrix B [5, 6] and the base address is 500. By using two method of arrange matrix in memory:
a. Draw logical structure and physical structure of the matrix B.
b. find the address of the element B [2, 3].

In case three Dimensional Arrays, memory-address of the element A[i,j,K] with dimension (NXMXR) is given by:

where:

R : number of levels
N: number of rows
M: number of column

In case of Row Major Order:

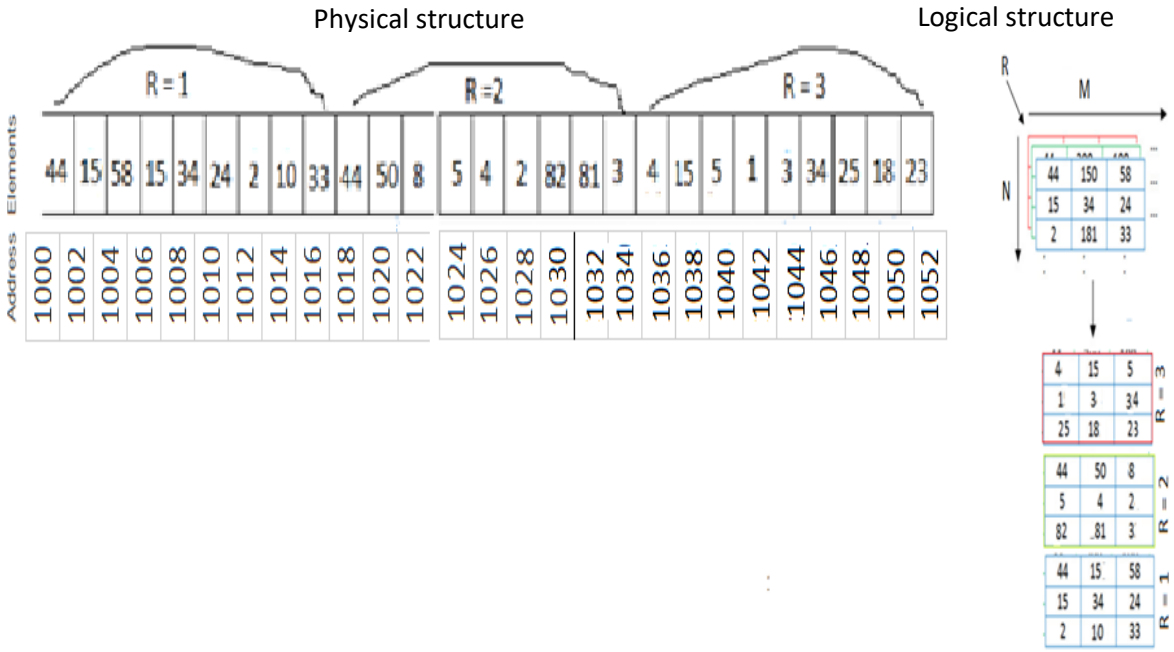
$$\text{Loc (A [i,j,k])} = \text{Loc (A [1, 1, 1])} + ([k-1]*N*M+ [i-1]*M+ (j-1))*W$$

In case of Column Major Order:

$$\text{Loc (A [i,j,k])} = \text{Loc (A [1, 1, 1])} + ([k-1]*N*M+ [j-1]*N+ (i-1))*W$$

Example : Suppose $A_{3 \times 3 \times 3}$(N=3, M=3 and R=3) integer array A and base address =1000 and number of bytes(w)=2. find the location of A [3,2,2] by using two method of arrange matrix in memory:

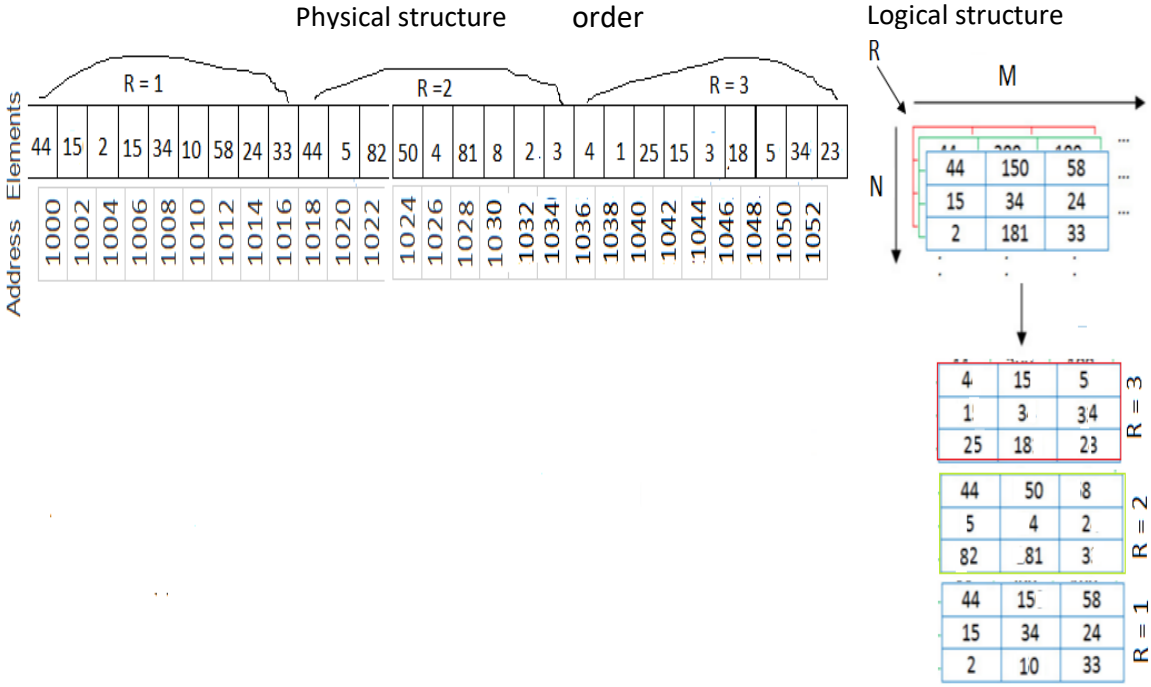
1. Row order



Loc(A[3,2,2])= 1000+2(3*3*(2-1)+3*(3-1)+2-1)

= 1000 + 2(9+6+1)

= 1032



Loc(A[3,2,2])= 1000+2(3*3*(2-1)+3*(2-1)+3-1)
= 1000 + 2(9+3+2)
= 1028

H.W.

You have the matrix A [5,3,2] ,by using two method of arrange matrix in memory:

a. Draw logical structure and physical structure of the matrix A.

b. find the address of the element A [2, 2,3].

In general, we can find out the location of any element in array (NXMXRXl) by using following formulas:

In case of Row Major Order:

LOC (A [i,j,k,l]) = Base (A) + w (NMR(l-1)+NM(k-1)+M(i-1)+j-LB)

In case of Column Major Order:

LOC (A [i,j,k,l]) = Base (A) + w (NMR(l-1)+NM(k-1)+N(j-1)+i-LB)

Triangular Matrix

A **triangular matrix** is a special kind of square matrix. A square matrix is called **lower triangular** if all the entries *above* the main diagonal are zero. Similarly, a square matrix is called **upper triangular** if all the entries *below* the main diagonal are zero.

Logical Structure

Upper Triangular Matrix

44	150	58
0	34	24
0	0	33

if i>j then $A_{ij} = 0$

Lower Triangular Matrix

44	0	0
58	34	0
150	24	33

if i<j then $A_{ij} = 0$

Physical Storage

Row Major order

Address	Elements
1000	44
1002	150
1004	58
1006	34
1008	24
1010	33

Address	Elements
1000	44
1002	58
1004	34
1006	150
1008	24
1010	33

Column Major order

Address	Elements
1000	44
1002	150
1004	34
1006	58
1008	24
1010	33

Address	Elements
1000	44
1002	58
1004	150
1006	34
1008	24
1010	33

We can find out the location of a[2,2]by using following formulas:

1. Upper triangular

In case of Row Major Order:

Loc (A [i, j]) = Base(A) + w(((i-1)*M - (i-1)*i/2) + (j-1))

Address	Elements
1000	44
1002	150
1004	58
1006	34
1008	24
1010	33

suppose Base (A)=1000 , i=2, j=2 and w=2
LOC (A [2,2]) = 1000 + 2*((2-1)*3-(2-1)*2/2+2-1)
= 1000 + 2(3-1+1)
= 1000 + 2*3
= 1000 + 6
= 1006

In case of Column Major Order:

Loc (A[i, j]) = Base(A) + w((j-1) *j / 2 + (i-1))

Address	Elements
1000	44
1002	150
1004	34
1006	58
1008	24
1010	33

suppose Base (A)=1000 , i=2, j=2 and w=2
LOC (A [2,2]) = 1000 + 2*((2-1)*2/2+2-1)
= 1000 + 2(1+1)
= 1000 + 2*2
= 1000 + 4
= 1004

2. Lower triangular

In case of Row Major Order:

Loc (A [i,j]) =Base(A) + w((i-1) * i /2 + ([j-1))

Address	Elements
1000	44
1002	58
1004	34
1006	150
1008	24
1010	33

suppose Base (A)=1000 , i=2, j=2 and w=2
LOC (A [2,2]) = 1000 + 2*((2-1)*2/2+2-1)
= 1000 + 2(1+1)
= 1000 + 2*2
= 1000 + 4
= 1004

In case of Column Major Order:

Loc (A [i,j]) = Base(A) + w(((j-1) *N - (j-1)*j/2) +(i-1))

Address	Elements
1000	44
1002	58
1004	150
1006	34
1008	24
1010	33

suppose Base (A)=1000 , i=2, j=2 and w=2
LOC (A [2,2]) = 1000 + 2*((2-1)*3 - (2-1)*2/2+2-1))
= 1000 + 2(3-1+1)
= 1000 + 2*3
= 1000 + 6
= 1006

Q: How determine the number of array elements?

Ans.: To determine the number of any array elements by applying the following equation:

$$\mathrm{M}\ \prod_{i=1}^{n}((U_i - L_i) + 1)$$

where:
n is dimensions of the array
U : upper bound for dimension i
L: lower bound for dimension i

Example1: Find the number of positions required to store the array: A [5]

$$\mathrm{M}\ \prod_{i=1}^{1}((U_1 - L_1) + 1)$$

=5-0+1=6

Example2: Find the number of positions required to store the matrix: A [5, 6]

$$\mathrm{M}\ \prod_{i=1}^{2}((U_1 - L_1) + 1) * ((U_2 - L_2) + 1)$$

(5-0+1)*(6-0+1)=6*7=42

Example3: Find the number of positions required to store the matrix:
A[2..5, 6...8]

$$\mathrm{M}\ \prod_{i=1}^{2}((U_1 - L_1) + 1) * ((U_2 - L_2) + 1)$$

= (5-2+1)*(8-6+1) = 4*3 =12

Arrays in programming languages

Arrays in C, C++

- we can declare an array by specifying its and size or by initializing it or by both. For Example:

```
int arr[10];      // Array declaration by specifying size
int arr[] = {10, 20, 30, 40}; //Array declaration by initializing elements.. Compiler
                              creates an array of size 4.

int arr[6] = {10, 20, 30, 40}; //Array declaration by specifying size and initializing
                                 elements
```

- Array elements are accessed by using an integer index. Array index starts with 0 and goes till size of array minus 1. Following are few examples.

```
int arr[5];
arr[0] = 5;
arr[2] = -10;
arr[3/2] = 2; // this is same as arr[1] = 2
arr[3] = arr[0];
```

- There is no index out of bound checking in C and C++:

```
int arr[2];
cout<< arr[3];
```

In C it is not compiler error to initialize an array with more elements than specified size.

```
int arr[2] = {10, 20, 30, 40, 50};
```

while in C++ , the program generates compiler error "error: too many initializes for 'int [2]'"

- In C/C++, initialization of a multidimensional arrays can have left most dimension as optional. Except the left most dimension, all other dimensions must be specified. For example:

```
int a[][2] = {{1,2},{3,4}};
```

Arrays in Java

Arrays in Java work differently than they do in C/C++. Following are some important point about Java arrays.

- In Java all arrays are dynamically allocated.
- Since arrays are objects in Java, we can find their length using member length. This is different from C/C++ where we find length using sizeof.
- A Java array variable can also be declared like other variables with [] after the data type.
- The variables in the array are ordered and each have an index beginning from 0.

In Java, all objects are dynamically allocated on Heap. This is different from C++ where objects can be allocated memory either on Stack or on Heap. In C++, when we allocate object using new(), the object is allocated on Heap, otherwise on Stack if not global or static.

In Java, when we only declare a variable of a class type, only a reference is created (memory is not allocated for the object). To allocate memory to an object, we must use new(). So the object is always allocated memory on heap.

One-Dimensional Arrays :

The general form of a one-dimensional array declaration is:

type var-name[];

OR

type[] var-name;

Example:

```
// both are valid declarations
int intArray[];  or  int[] intArray;
float floatArray[];
double doubleArray[];
char charArray[];
```

Although the above first declaration establishes the fact that intArray is an array variable, **no array actually exists**. It simply tells to the compiler that this(intArray) variable will hold an array of the integer type. To link intArray with an actual, physical array of integers, you must allocate one using **new** and assign it to intArray.

Instantiating an Array in Java

When an array us declared, only a reference of array is created. To actually create or give memory to array, you create an array like this: The general form of *new* as it applies to one-dimensional arrays appears as follows:

var-name = new type [size];

Here, *type* specifies the type of data being allocated, *size* specifies the number of elements in the array, and *var-name* is the name of array variable that is linked to the array. That is, to use *new* to allocate an array, **you must specify the type and number of elements to allocate.**

Example:

```
int intArray[];                    //declaring array
intArray = new int[20];            // allocating memory to array
        OR
int[] intArray = new int[20];      // combining both statements in one
```

Note :

1. The elements in the array allocated by *new* will automatically be initialized to **zero** (for numeric types), **false** (for boolean), or **null** (for reference types).
2. Obtaining an array is a two-step process. First, you must declare a variable of the desired array type. Second, you must allocate the memory that will hold the

array, using new, and assign it to the array variable. Thus, **in Java all arrays are dynamically allocated.**

Array Literal

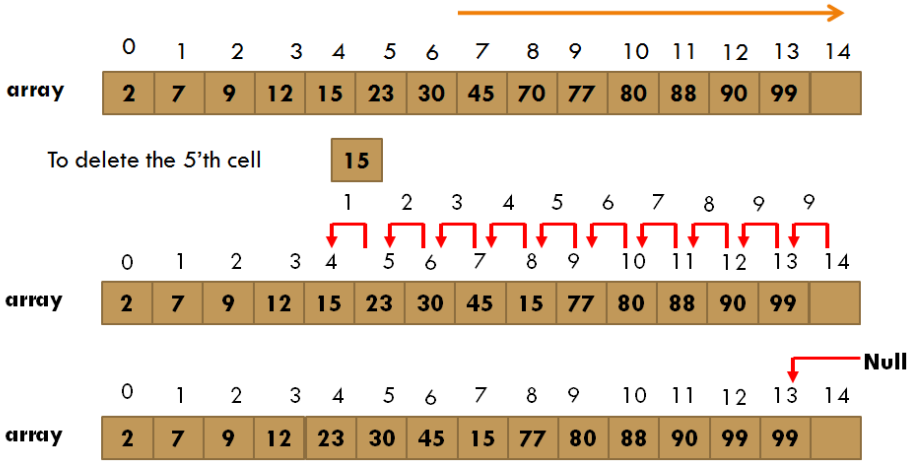
In a situation, where the size of the array and variables of array are already known, array literals can be used.

int[] intArray = new int[]{ 1,2,3,4,5,6,7,8,9,10 }; // Declaring array literal

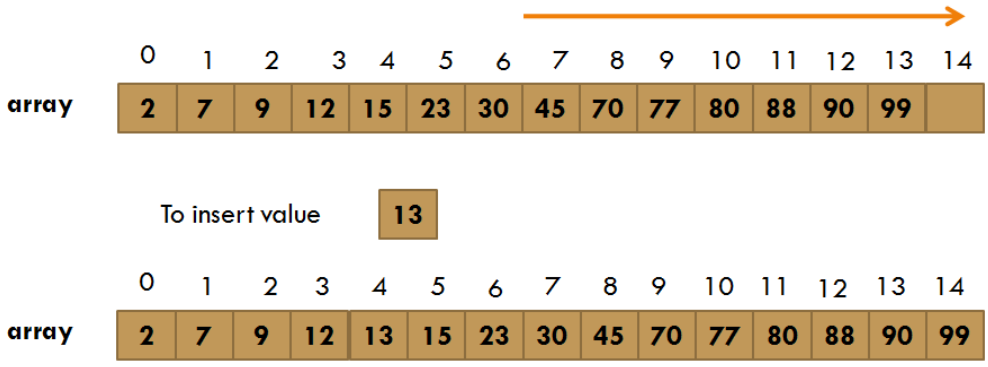
- The length of this array determines the length of the created array.
- There is no need to write the new int[] part in the latest versions of Java

Q:How to delete and insert an element from array?

Ans.: Insertion and deletion at particular position is complex, it require shifting as in the following examples.



Insert item to sorted array



Q: Write a program to insert item to sorted array.

Q: What happens if we try to access element outside the array size?

Ans: Compiler error, indicate that array has been accessed with an illegal index. The index is either negative or greater than or equal to size of array.

```
class example{
   public static void main (String[] args){
      int[] arr = new int[2];
      arr[0] = 10;
      arr[1] = 20;
      for (int i = 0; i <= arr.length; i++)
         System.out.println(arr[i]);}}
```

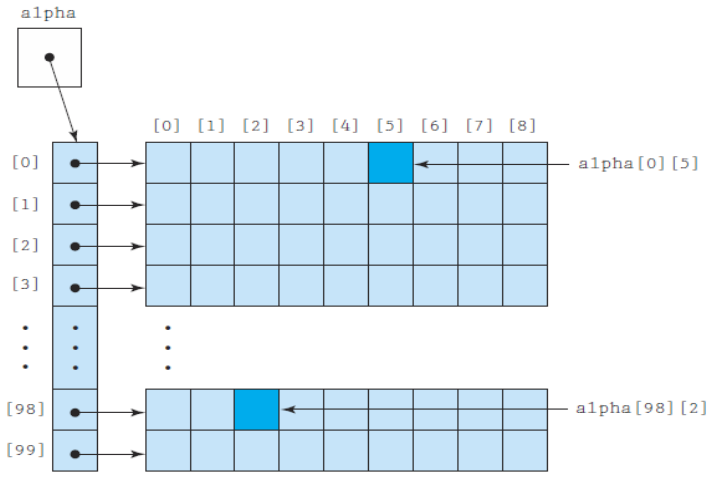
Two-Dimensional Arrays :

In Java Multidimensional arrays are **arrays of arrays** with each element of the array holding the reference of other array. For example, 2D array is a 1D array of references to 1D arrays, each of these 1D arrays (rows) can have a different length, this 2D array is called **"Jagged array".**

Examples of declare two-dimensional array:

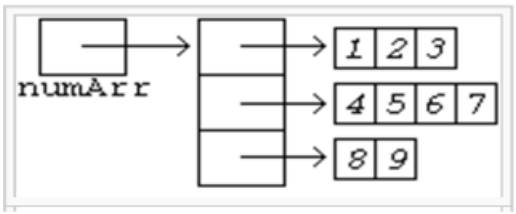
int[][] intArray = new int[10][20]; //a 2D array or matrix

int[][][] intArray = new int[10][20][10]; //a 3D array



Example: In the following memory layout of a jagged array numArr.

int[][] numArr = { {1,2,3}, {4,5,6,7}, {8,9} };



And jagged arrays can be created with the following code:

```
int [][]c;
c=new int[2][];
c[0]=new int[5];
c[1]=new int[3];
```