Data Structures and Algorithm Analysis

2

Dr.Syed Asim Jalal
Department of Computer Science
University of Peshawar

Character Representation

- We can also interpret some series of 1s and 0s as characters/alphabets.
- We assign a series of 1s and 0s some English characters.
- Generally, we can represent 2^n characters, in a scheme that uses *n* bits to represent characters.
  - e-g: 8 bits for each characters would represent 256 characters.
- How many bits needed to represent English characters????
  - 26 CAPITAL case LETTERS
  - 26 Capital + 26 Lower case letter
  - 26 Capital + 26 Lower + Digits + Special Characters.

Character Representation

- Different schemes for representation of characters representation have been proposed.
  - ASCII is one such representation where each 7 bits are used to represent English characters.
  - e.g:
    - A is represented by 01000001
    - B is represented by 01000010

- Some characters representation schemes are
  - ASCII: American Standard Code for Information Interchange
  - EBCDIC: Extended Binary Coded Decimal Interchange Code
  - Unicode: Universal Coding

- ASCII
  - American Standard Code for Information Interchange
  - It was designed in the early 60's, as a standard character set for computers and electronic devices.
  - Representation of **English letters** and some other characters
  - Each character is represented using 7 bits, while one bit is used for **parity checking**.
  - 7 bits could represent **128 characters**

Representation of some characters

ASCII Code: Character to Binary

0	0011 0000	O	0100 1111	m	0110 1101
1	0011 0001	P	0101 0000	n	0110 1110
2	0011 0010	Q	0101 0001	o	0110 1111
3	0011 0011	R	0101 0010	p	0111 0000
4	0011 0100	S	0101 0011	q	0111 0001
5	0011 0101	T	0101 0100	r	0111 0010
6	0011 0110	U	0101 0101	s	0111 0011
7	0011 0111	V	0101 0110	t	0111 0100
8	0011 1000	W	0101 0111	u	0111 0101
9	0011 1001	X	0101 1000	v	0111 0110
A	0100 0001	Y	0101 1001	w	0111 0111
B	0100 0010	Z	0101 1010	x	0111 1000
C	0100 0011	a	0110 0001	y	0111 1001
D	0100 0100	b	0110 0010	z	0111 1010
E	0100 0101	c	0110 0011	.	0010 1110
F	0100 0110	d	0110 0100	,	0010 0111
G	0100 0111	e	0110 0101	:	0011 1010
H	0100 1000	f	0110 0110	;	0011 1011
I	0100 1001	g	0110 0111	?	0011 1111
J	0100 1010	h	0110 1000	!	0010 0001
K	0100 1011	i	0110 1001	'	0010 1100
L	0100 1100	j	0110 1010	"	0010 0010
M	0100 1101	k	0110 1011	(	0010 1000
N	0100 1110	l	0110 1100	)	0010 1001
				space	0010 0000

EBCDIC and Unicode

- EBCDIC:
  - Extended Binary Coded Decimal Interchange Code
  - EBCDIC uses 8 bits to represent characters
  - 8 bits could represent 256 characters
  - It was used mainly on IBM mainframe and IBM midrange computer operating systems

- Unicode
  - Unicode character coding was developed to represent character set of many different languages
  - Unicode using **16 bits encoding**
  - The latest version of Unicode cover over **128000** characters of **over135 languages** and many special symbols.

So from the discussion of data representation we can see that a sequence of 0s and 1s mean nothing by itself. The important thing is how we assign meaning to any sequence of of 1s and 0s and later interpret this sequence.

Some times we assign a numeric value
Some times we assign a signed number
Some times Alphabets

Data Types

- A Data Type describes a way of interpreting a bit pattern in the memory.
- A Data Type defines internal representation of data in the memory.
- It also specifies a set of operations on that data type.
- It also defines the Hardware or Software implementation of the data type
  - Hardware implementation: Implementation by processor.
  - Software implementation: Implementation by program.

Some Terminologies

- Data
  - Data are any values or set of values
- Data Item
  - Data item is a single unit of values
    - Name, Age, Gender
- Group Item
  - Data Items that are divided into sub-items are called group items
    - E.g. Name divided into First Name and Family Name
- Elementary Items
  - Data items not divided into sub-items
    - E-g: Age.

Some Terminologies

- Entity
  - It is something that has certain attributes. Each attributes has a value or values.
  - For example:
    - Student is an entity with attributes, Name, Age, Gender, Date of Birth, etc.
    - Each attributes has some value

- Entity Set
  - Collection of all entities with same attributes
  - Collection of all instances of an Entity

Some Terminologies

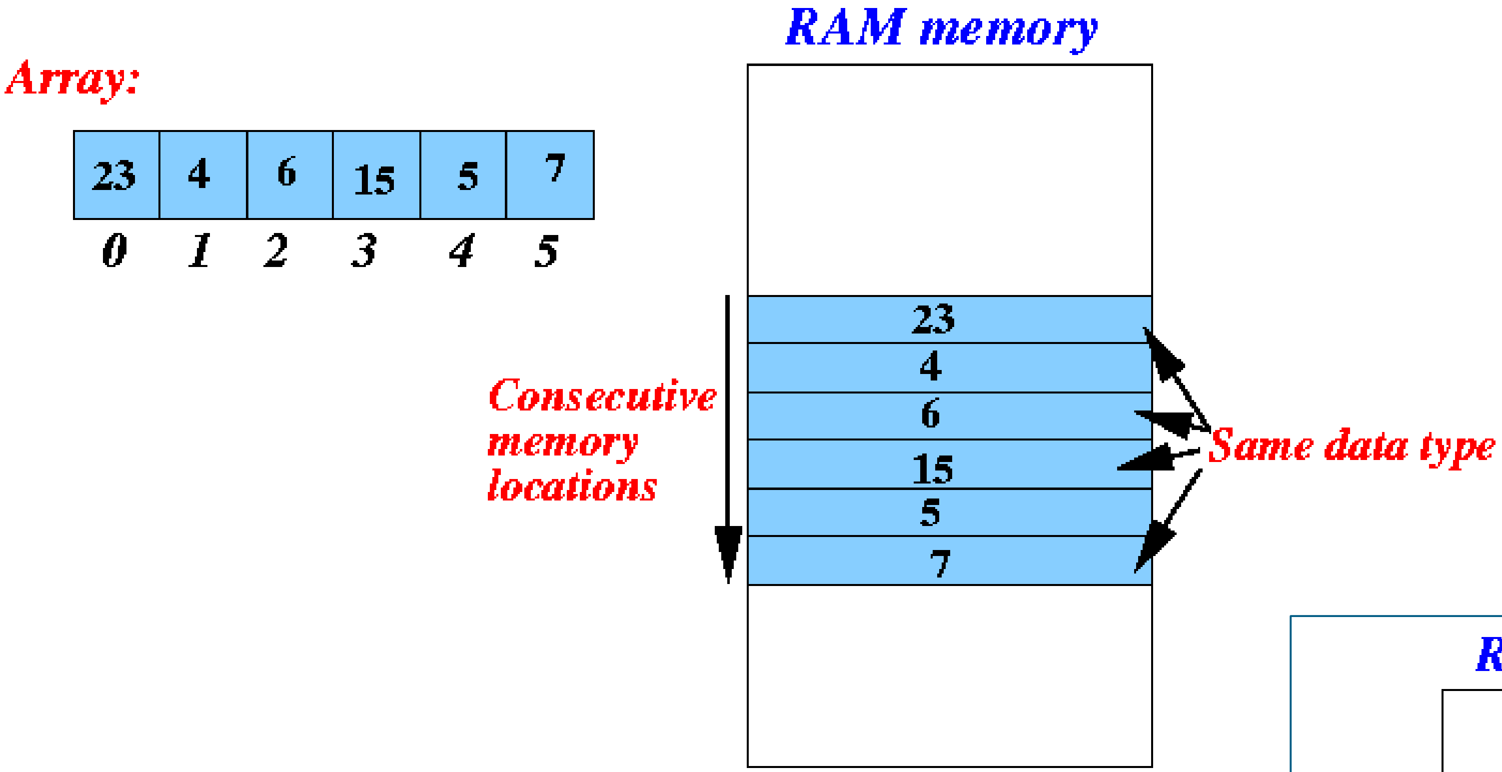
- Collection of data organized into Fields, Records and Files.
- Field
  - Field is a single unit of information representing a single attribute of all entities.
  - e-g: Name Field.
- Record
  - It is collection of Field of values of one entity
- File
  - It is a named collection of all records representing all entities.

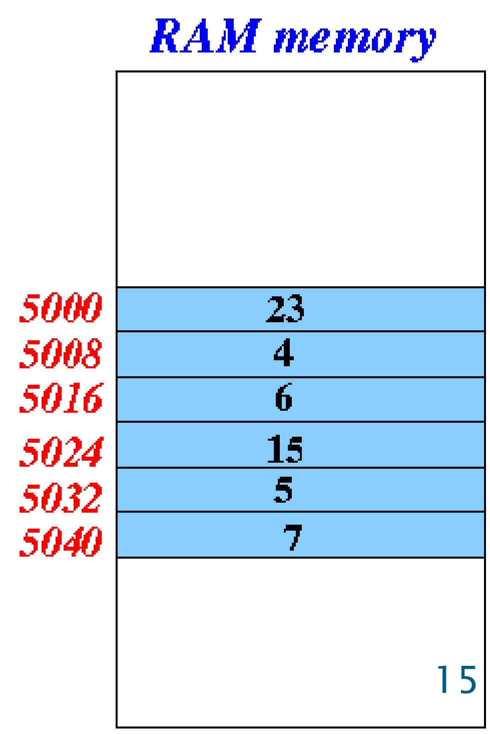
Arrays

Array

- Array is a composite or non-primitive data type, that is, it is made up of simpler data types.
  - Array is a data structure that organizes a collection of data of the *same data type* using consecutive memory cells.
  - Array is a list of finite number *‘n’* of homogeneous data elements, where:
    - The elements of the array are stored in successive memory locations
    - The elements of the array are referenced by an index. Index values are *‘n’* consecutive numbers.

- Arrays exists in most programming languages and operations of this data structure are already implemented by those programming languages.


Array:
23
4
6
15
5
7
0
1
2
3
4
5
RAM memory
23
4
6
15
5
7
Consecutive memory locations
Same data type


RAM memory
5000
5008
5016
5024
5032
5040
23
4
6
15
5
7

- Each array element occupies the same number of memory cells (bytes)
- Array data structure is used when the number of elements are fixed.
- Operations like traversal, searching and sorting can be easily performed on Array.
- The number of elements in an array is called the Length or Size of the Array
- The size of array is specified at creation or declaration of the array

- Index consists of integers 0,1,2,3…,n
- Index is mostly represented by a number in brackets after name of the array, e-g. x[0], x[1], x[2], x[3], x[4], x[5]
- The name of the array is a pointer to first value. That is, name of an array stores address of first memory.
- Arrays can be
  - One dimensional array
    - Array with one index. A[5]
  - Two dimensional array
    - Array with two indexes. A[5][3]
  - Or n-dimensional

One dimensional array

- One dimensional array is the **simplest** form of an Array
- One dimensional array may be defined as a finite ordered set of homogeneous elements
  - Finite means limited number of elements
  - Homogeneous means all elements are of the same type
  - Ordered means that the elements are arranged such there exists element at index 0, 1, 2 and so on.

- In C language, we declare a one-dimensional array as the following
  - `int arrayName[100];`
    - Name of the array is ‘arrayName’
    - Total number of elements is 100
    - Each elements is an integer
    - ‘arrayName’ is a pointer and stores address of the memory location of the first value
- The smallest index of an array is called Lower Bound, the largest index is called Upper Bound

Size of array = Upper Bound – Lower Bound + 1

- Reading a value
  - a[i] returns the value stored at index i
  - The first value is referenced by index 0, that is, a[0]
- Assigning a value
  - a[i] = x;
  - Value x is stored at location i
  - Before any value is assigned to any location, the value of that location is undefined.

Addressing in one-dimensional array

- As size of each element in an array is same, the computer, therefore, does not need to know address of each element in advance.

- Address of each element can be calculated during run time using **index number** and the **Base Address** of the array.
  - The base address of the array is always known and is represented by the name of the array.

- Address calculation:
  - The address of the first location of an array B is called base address of **B**, and is denoted by ***Base(B)***
  - Suppose ***esize*** is the memory *size* of each *element*.

- Then address of the B[0] element is ***Base(B)***
- Address of B[1] element would be ***Base(B) + esize***
- Address of B[2] would be ***Base(B) + 2 * esize***
- So the general expression to reference address of ***B[i]*** would be ***Base(B) + I * esize***

- What will happen if index of an array starts with 1 instead of 0 in any programming language?
- The formula to access B[i] becomes

<u>Base(B)+ (i – 1) * esize</u>

Two dimensional (2-D) array

- A two dimensional array has two indexes to address each element, for example, **B[2][4]**
  - First index represent **Row** and
  - second index represent **Column** number.

- It has rows as well as columns. Such an array can be considered as array of arrays.

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

- Total number of rows and columns is called range of that dimension

- Thinking in 2 dimensions is convenient for programmers in many situations.

  - In situations where any set of values that are dependent on two inputs.
  - For example, a departmental store that has 20 branches each selling 30 items.
    - int sales[20][30];
    - **sales[i][j]** would represent sales of item j in branch i.

- The problem in a 2-Dimensional array is that it is only a logical data structure, because physical hardware does not have such facility (i.e. memory is linear and sequential addresses).
- A 2-D array must be stored linearly in the memory, therefore, a method is needed that would convert a Row and Column indexes of 2-D array in a linear memory addresses.

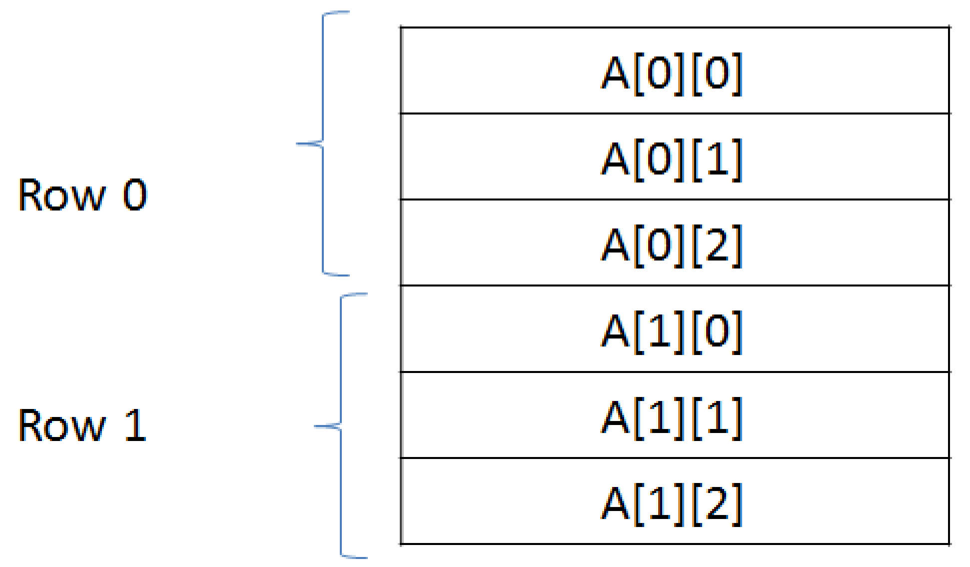
Implementing a 2-D array

- We have two major approaches for mapping from 2-D logical space to 1-D physical space
- Two approaches
  - Row Major order
  - Column Major order

Row-Major Order

- The first row of the array occupies the first set of memory locations reserved for the array, the second row occupies the second set, and so on.

- For example, A[2][3] would be represented as:



Representation of A[2][3] in Row-major Order

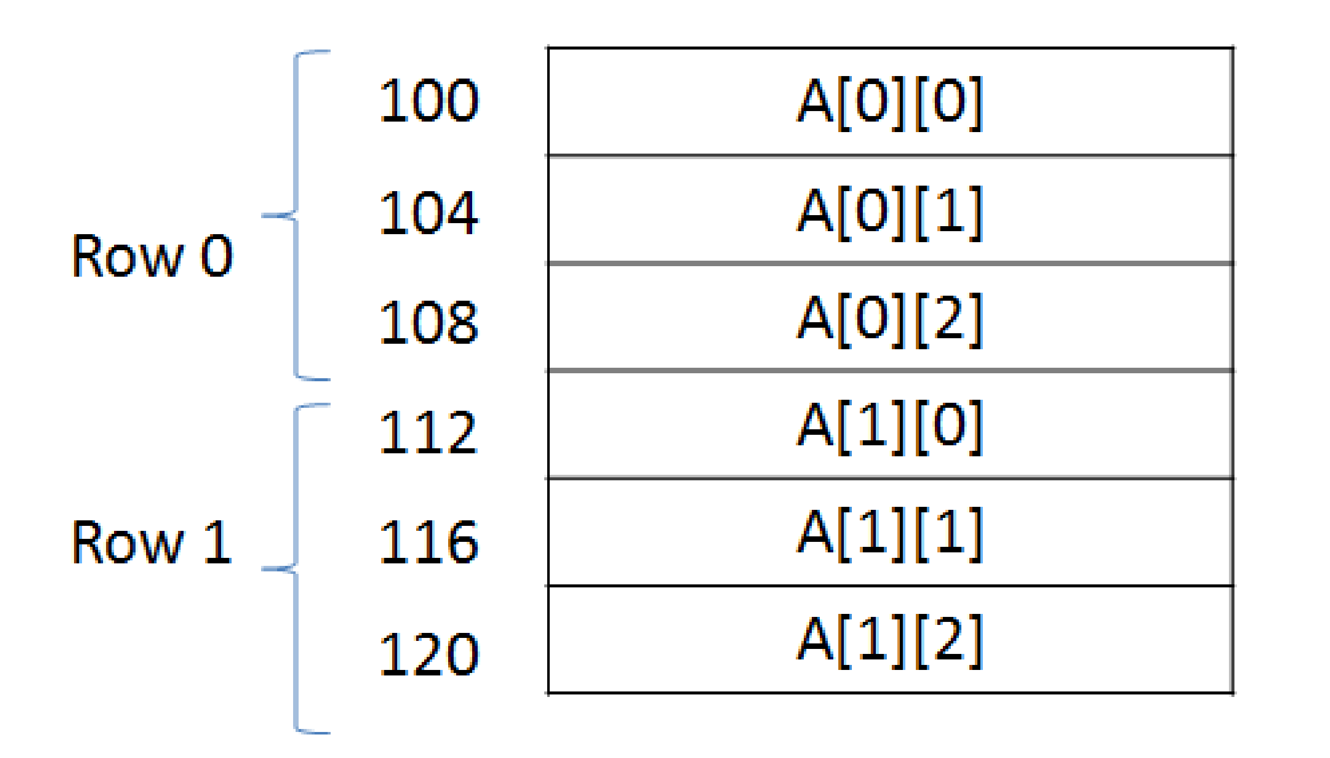
- Finding address of an element in Row-Major Order
  - Suppose **int A[Rows][Columns]** is stored in row-major order with base address *base(A)* and element size *esize*.
  - Then the address of the element **A[r][c]** can be calculated by calculating the address of the first element of *row* ***r*** and adding the quantity ***c * esize***
  - The address of first element of row **r** is *base(A)+r * cols * esize*
  - Therefore the address of **A[r][c]** is

    *base(A)+ (r * cols) * esize* ***+ c * esize***

- Or simplifying the expression
- We get the address of *A[r][c]* as:

$$base(A) + (r * cols + c) * esize$$

- Example:
  - Address of A[r][c] = base(A)+(r*cols + c) * esize
  - Here base(A)=100, rows=2, cols=3, esize=4



Array Initialisation Algorithm

– Algorithm for assigning array values

Suppose LB = LowerBound, UP=UpperBound and Array is name of the Array

1. Initialise counter c to lower bound, c = LB
2. Repeat step 3 and 5 while c < = UB
3. **value** = input new value
4 Assign **value** at index c, Array[c] = **value**
5 Increment counter: c = c + 1
6. Exit

Array Traversal Algorithm

– Traversal means access each element of the array once for process.

Suppose LB = LowerBound, UP=UpperBound and Array is name of the array to traverse.

1. Initialise counter, c = LB
2. Repeat step 3 and 4 while c < = UB
3. visit and process Array[c]
4. Increment counter: c = c + 1
5. Exit