Arrays

Data Structures: A Programming Approach with C, PHI,
Dharmender Singh Kushwaha & A.K.Misra

Outline

- Introduction
- One dimensional array
- Basic Operations on Array
- Multi Dimensional Array
- 3-Dimensional Array
- Summary

Introduction

- An array is a collection of elements of the same type
- Each element of the array is always stored in consecutive memory location.
- Each element is accessed using a subscript or index.
- int arr[100]; declares an array of 100 integers
- Indices for above arrays are 0 to 99.

Introduction

- Array name is actually a pointer to the first location of the memory block allocated to the name of the array.

- Examples

  int arr[5]={1,2,3,21,11}

  array arr is initialized with 5 values.

Introduction

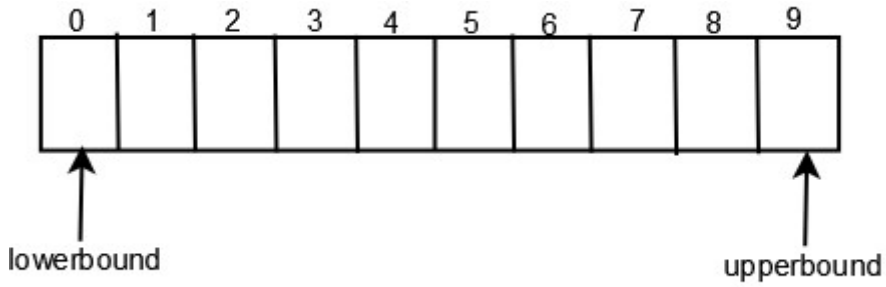
- Array name is actually a pointer to the first location of the memory block allocated to the name of the array.
- It can be initialized during declaration time only.
- Since there is no bound checking for array in C / C++, we have to check for upper bound of array

Example 1: Illustrating initialization of array

```
int[] marks;                    // marks is a reference name given to an array of integers

marks = int[20];                // allocates OS storage for 20 integers

int[] marks =  int[20];         // combines the above into one line

marks[2] = 91;                  // gives the array element index 2 the value 91
```

One dimensional array

- A one dimensional array requires **o**nly one subscript specification.
- int A[10];



- The reference name given to the array is A and ‘i’ is the subscripted variable that is the index in the array.

One dimensional array

- ‘i' can take values from 0 to 9.
- Size of the array is calculated as

Size = (upperbound – lowerbound) + 1

- Calculating memory requirement

Size in byte = size of array * size of (element type)

One dimensional array

float val[10];

size = 10 * sizeof (float)

= 10 * 4

= 40 bytes

Sizeof operator gives the bytes required by that particular data type.

Implementation of 1- D Array in Memory

- Base address of the array is the address of first element of the array.
- Address of k^{th} element **of the array a[ ] is**

a[k] = B + (S * k)

B=Base address of array.

S=Size of each element of array.

k=Index of the element.

Basic Operations on Array

- Some common operation types are:
  - Traversal of array
  - Inserting an element in the array
  - Deleting an element from the array
  - Merging two arrays

Traversal of Array

Traversal of an array is performed to:

- Read and access all elements one by one.
- To access the elements or
- To perform any other operation.

Pseudocode Traversal

- Let the array to be traversed be named as ‘arr’ containing ‘n’ elements such that:
  - element at position 0 is lowerbound ‘lb’ of the array and
  - the element at index position n be the upperbound ‘ub’ of the array.

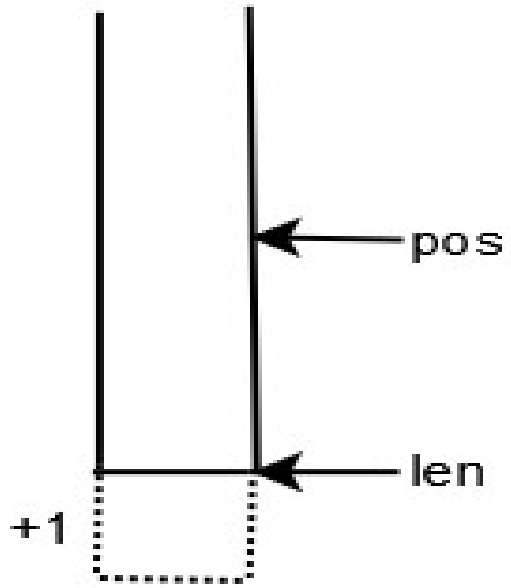
Pseudocode Traversal

```
Initialize counter i
Set i at lb
Repeat
    for i=lb to ub
    visit element
    display arr[i]
end of loop
Exit
```

Inserting an element in 1-D Array

- Consider an array arr whose length is *len and the elements in this array are stored from location 0* to *len.*
- *In order to insert an element 'value' in the array at some location LOC, we have to shift* all the elements right from index *pos till len by one.*
- *The index of LOC is less than the index of len of the array.*
- *This will vacate one memory location at pos so as to insert 'value'. Let Insert be* the function that does this job and the pseudo-code for this is given here.

Inserting an element in 1-D Array



Data Structures: A Programming Approach with C, PHI,
Dharmender Singh Kushwaha & A.K.Misra

Pseudocode: Insert (arr, len, pos, value)

Initialize the value of i=length of the array arr.

Repeat

    for i = len down to pos

        set arr[i+1]=a[i]   // shift the element down by 1 position

        end of loop

    Insert element at required position

        set arr[pos]=value

        set len = len + 1   // Reset length of array

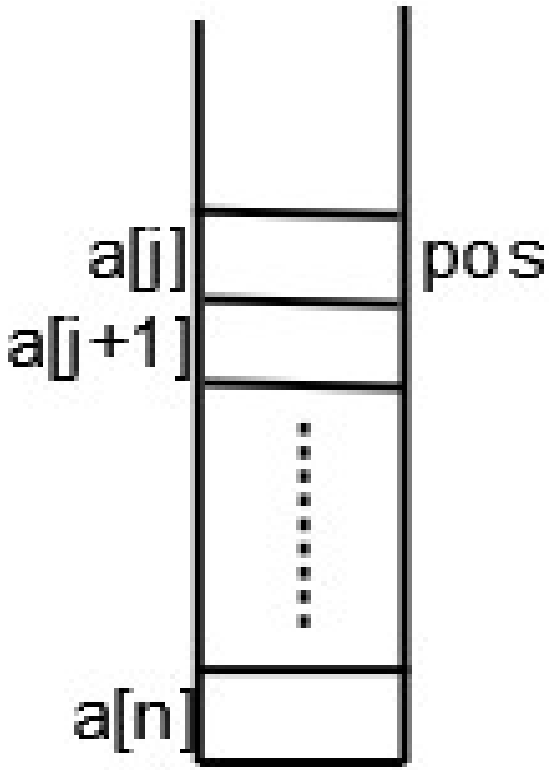
    Display new list

End

Deleting an Element from 1-D Array

- Deletion of an element placed at the end of an array poses no problem.
- If last element is deleted, it simply reduces array length by one.
- Deleting an element placed in the middle of an array requires shifting all elements as to fill the space emptied by the just deleted element.

Deleting an Element from 1-D Array



Data Structures: A Programming Approach with C, PHI,
Dharmender Singh Kushwaha & A.K.Misra

Pseudocode Delete (arr, pos, n)

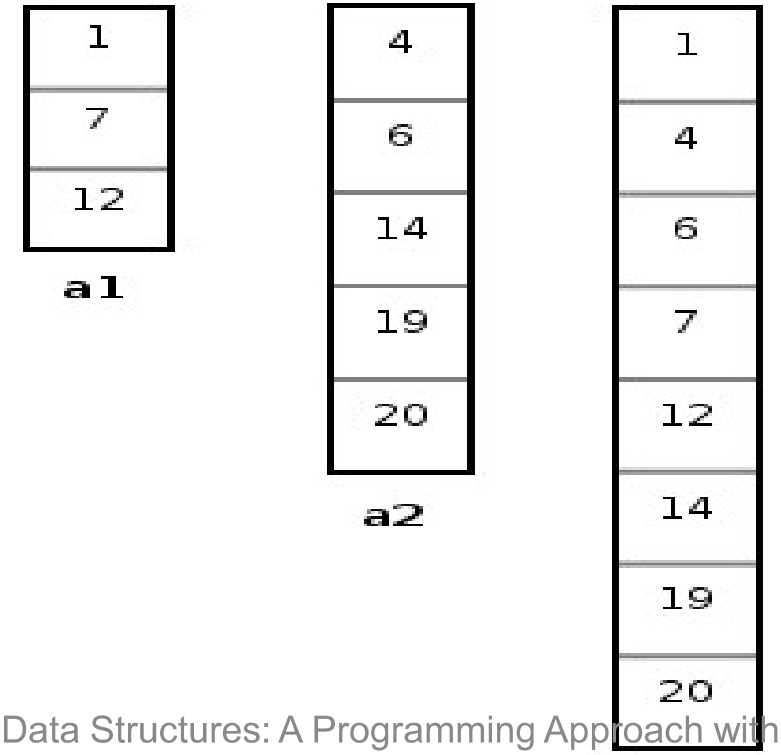
```
Set item =a [pos]      //store the element to be deleted in
                       a variable
Repeat
    for j = pos to n-1 //since there are n elements in the array
    set a [j] = a [j+1] //shift elements one position leftwards /
                       upwards
end of loop
Reset number of elements in the array as n=n-1
End
```

Merging two Arrays

- There are several applications where we have to merge the elements of two different arrays into one array.
- The elements in the two given arrays arr1 and arr2 may be in some sorted order.
- In the given pseudocode, the elements are arranged in ascending order and the resultant merged array arrmerge also has all its elements arranged in the ascending order.

Merging two Arrays

- The resultant merged array a3 also has all its elements arranged in the ascending order.



Merging two Arrays

- Create three array arr1[10], arr2[10] & arr3[20].
- Enter or arrange the element in ascending order in arr1 & arr2.
- Initialize the lower bounds lbarr1, lbarr2 & lbarr3 to 0th position of the 3 arrays.

Check if

arr1[lbarr1] < = arr2[lbarr2]

Merging two Arrays

If yes, assign elements of arr1 to arr3

```
    arr3 [lbarr3]= arr1[lbarr1]          // First element of
                                         arr1 copied to arr3
    lbarr1++ //set lbarr1 = lbarr1 + 1
    lbarr3++ //set lbarr3 = lbarr3 + 1
else
    arr3[lbarr3] = arr2[lbarr2]          // assign
                                         elements of arr2 to arr3
```

Merging two Arrays

lbarr2++ //set lbarr2 = lbarr2 + 1

lbarr3++ //set lbarr3 = lbarr3 + 1

end of loop

Repeat the above loop for remaining elements of arr1, arr2 & arr3.

Display elements of arr3

End.

Character Array

```
#include <stdio.h>
main()
{
    static char chararr1[] = {'H','e','l','l','o'};
    static char chararr2[] = "Hello";
    printf("%s\n", name1);
    printf("%s\n", name2);
}
```

Sample Program Output

Helloxghifghjk (some garbage value)

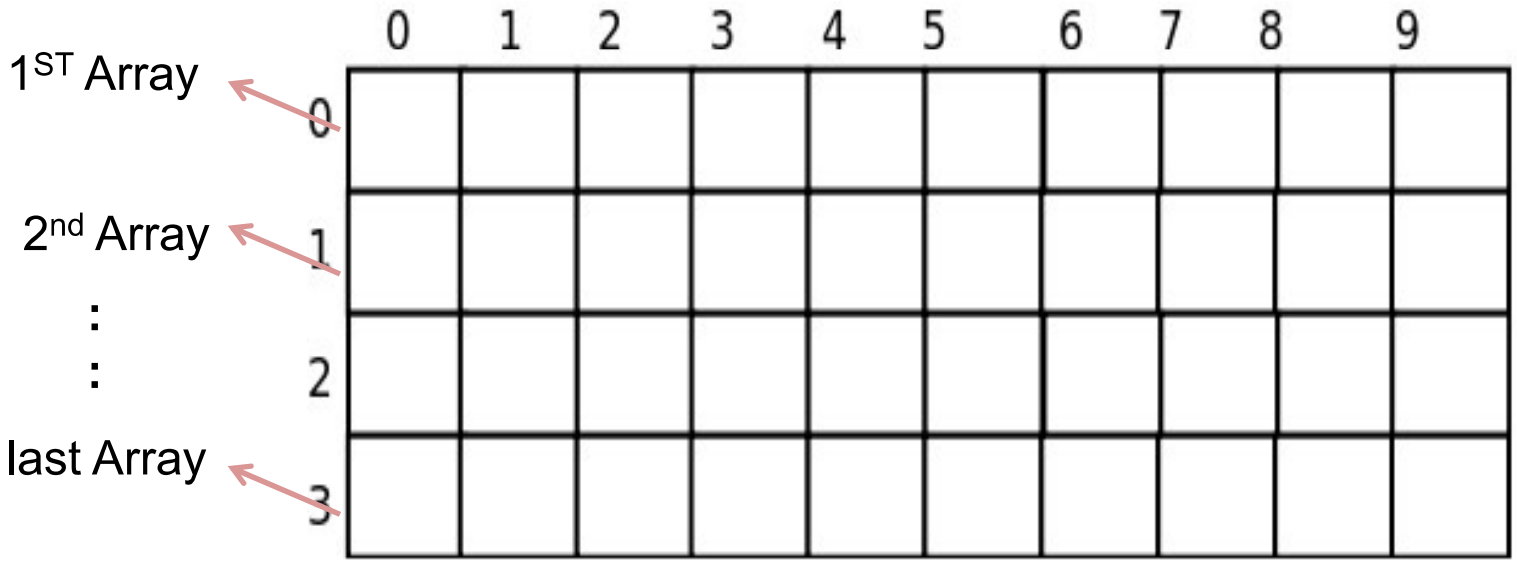
Hello

Character Array

- The difference between the two arrays is that *chararr2 has a null placed at the end of the string,* ie, in *chararr2[5], while chararr1 doesn't.*
- *This often results in garbage characters being printed* on the end.
- To prevent this, we insert a null at the end of the *chararr1 array.*
- *The initialization* now changes to:
  - *static char chararr1[] = {'H','e','l','l','o','\0'};*

Multi Dimensional Array

- If we place identical arrays adjacent to each other, what we have is a multidimensional array.



Data Structures: A Programming Approach with C, PHI, Dharmender Singh Kushwaha & A.K.Misra

Multi Dimensional Array

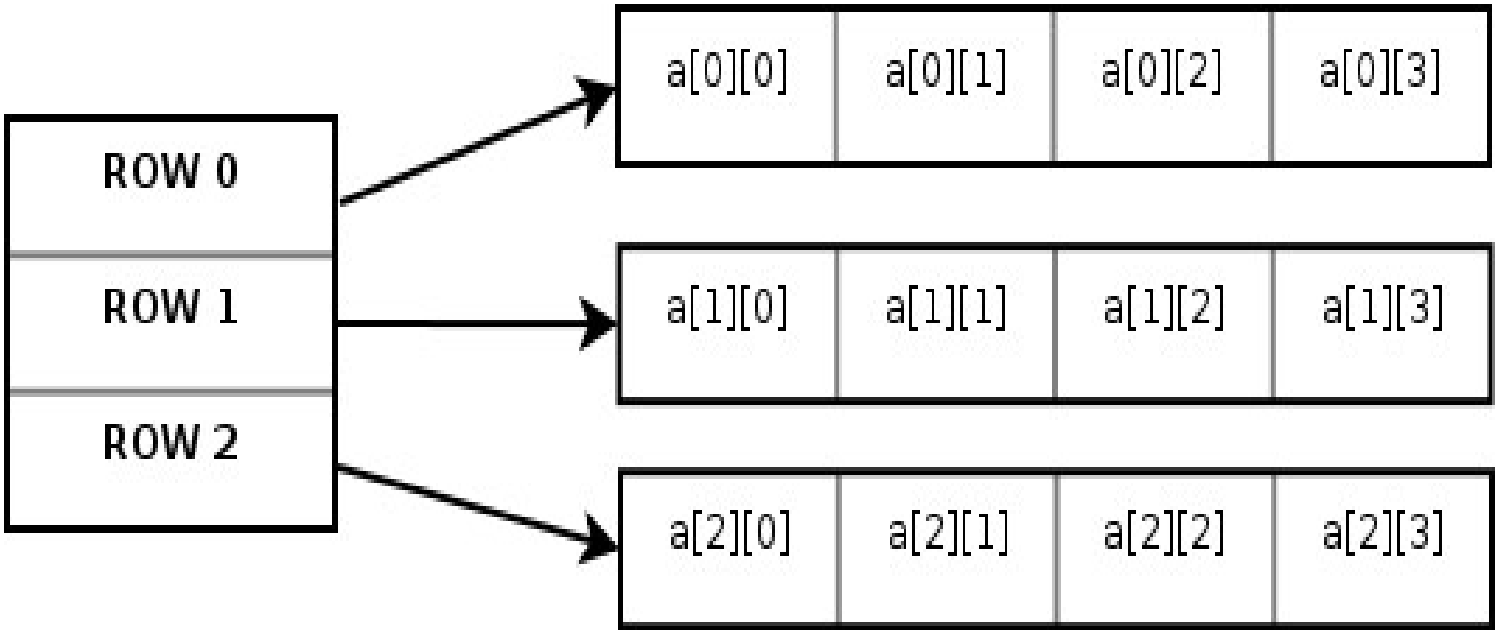
- All elements are homogeneous.
- A pair of indices specifies the desired element in a 2-dimensional array.

Example:

- int a[3][4]
- First index i.e. 3 represents the number of rows and second index i.e. 4 represents number of columns in array.

Multi Dimensional Array

- A[0][0] is the first and a[2][3] is the last element of the array.
- An element a[i][j] refers to an element that is in i^{th} row and j^{th} column.



Data Structures: A Programming Approach with C, PHI, Dharmender Singh Kushwaha & A.K.Misra

Multi Dimensional Array

- In memory, elements of a 2-dimensional array are stored in contiguous memory locations.

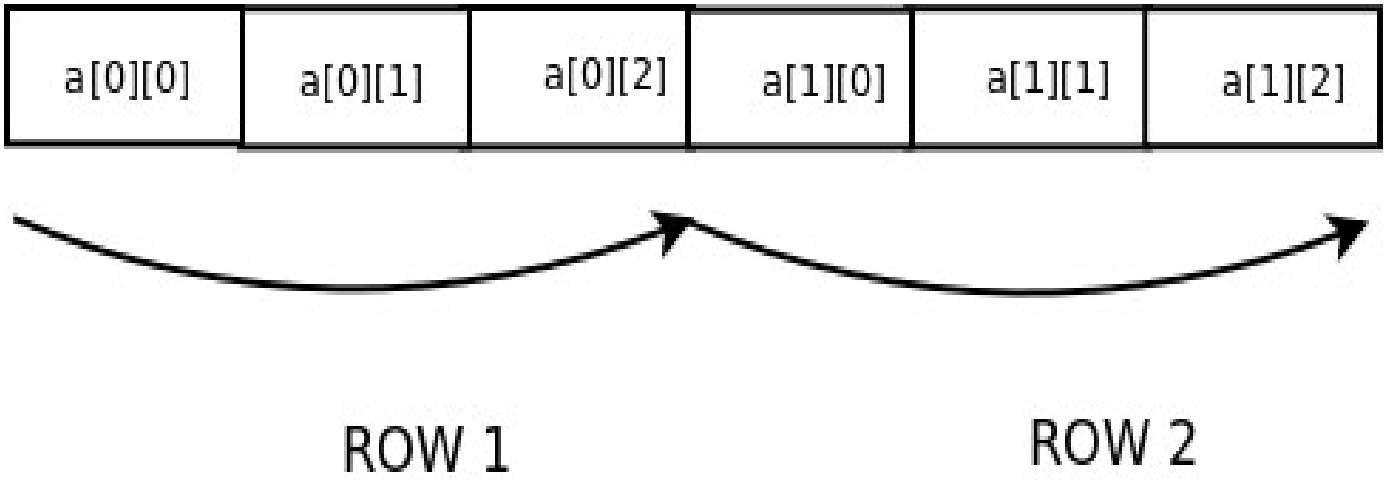
- 2 dimensional array can be initialized by specifying the position and value as follows:

  - A[1][3]=32 inserts 32 in 1st row and 3rd column.

Implementation of 2-D Array

Row major implementation

- Elements of an array are stored and accessed row wise.
- All the elements of row 1 are stored first and then the elements of row 2 and so on.



Implementation of 2-D Array

Address of element a[i][j] = [base address] + [number of byte required by each element * (number of column (i - lb of row) + (j - lb of column))]

or

Add of a[i][j]=B + nb [n * (i - lbr) + (j - lbc)] where:

B = base address of the array

nb = number of bytes occupied by each element of the array

i ,j = index of element at a[i,j] whose address is being computed

lbr = lower bound of row

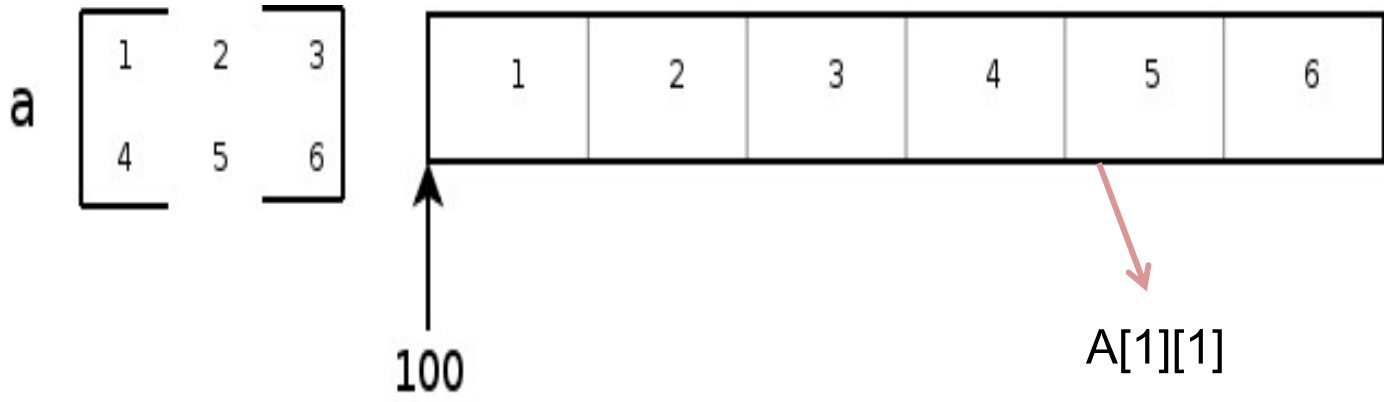
lbc = lower bound of column

m = number of rows in the array

n = number of columns in the array

Implementation of 2-D Array

- Calculate address for a[1][1].
- Here B=100, nb=1 byte, i =1, j =1,
- lbr = 0, lbc = 0,n = 3
- Add a[1][1]=100 + 1 [3 (1 – 0) + (1 - 0)] $=104$

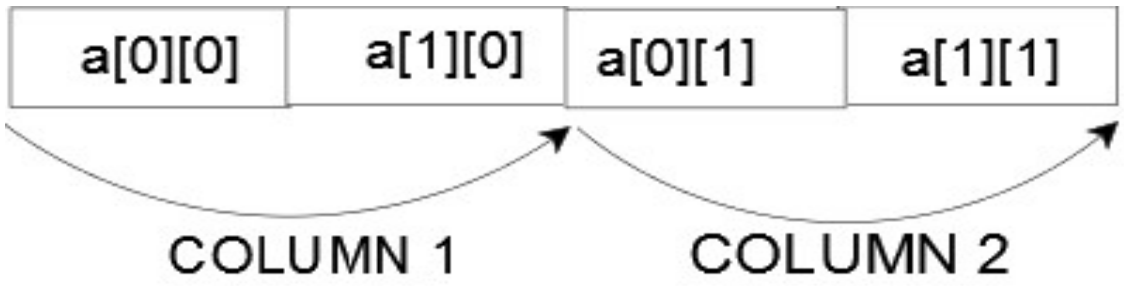


Data Structures: A Programming Approach with C, PHI, Dharmender Singh Kushwaha & A.K.Misra

Implementation of 2-D Array

Column major implementation

- Elements of an array are stored and accessed column wise.
- All the elements of column 1 are stored first and then the elements of column 2 and so on.



Implementation of 2-D Array

Address of element a[i][j] = [base address] + [number of byte required by each element * (number of rows (j - lb of col) + (i - lb of row))]

or

add of a[i][j]= B + nb [m * (j- lbc) + (i - lbr)] where:

B = base address of the array

nb = number of bytes occupied by each element of the array

i,j = index of element at a[i,j] whose address is being computed

lbr = lower bound of row

lbc = lower bound of column

m = number of rows in the array

Example

- Given a 2-D array arr[-10..20, 10..35] where each elements requires 1 bytes of storage and the array is column major implementation. Calculate the address of element stored at arr[0, 30].
- The base address of the given array is 1010.

Example

- B = 1010
- nb = 1
- i =0
- j =30
- lbr = -10
- lbc = 10
- m = 31

Example

Add of a[0][30] = 1010 + 1[31*(30 – 10) + (0 –(- 10))]

= 1010 + [31*(20) + 10]

= 1010 + [620 + 20]

= 1010 + 640

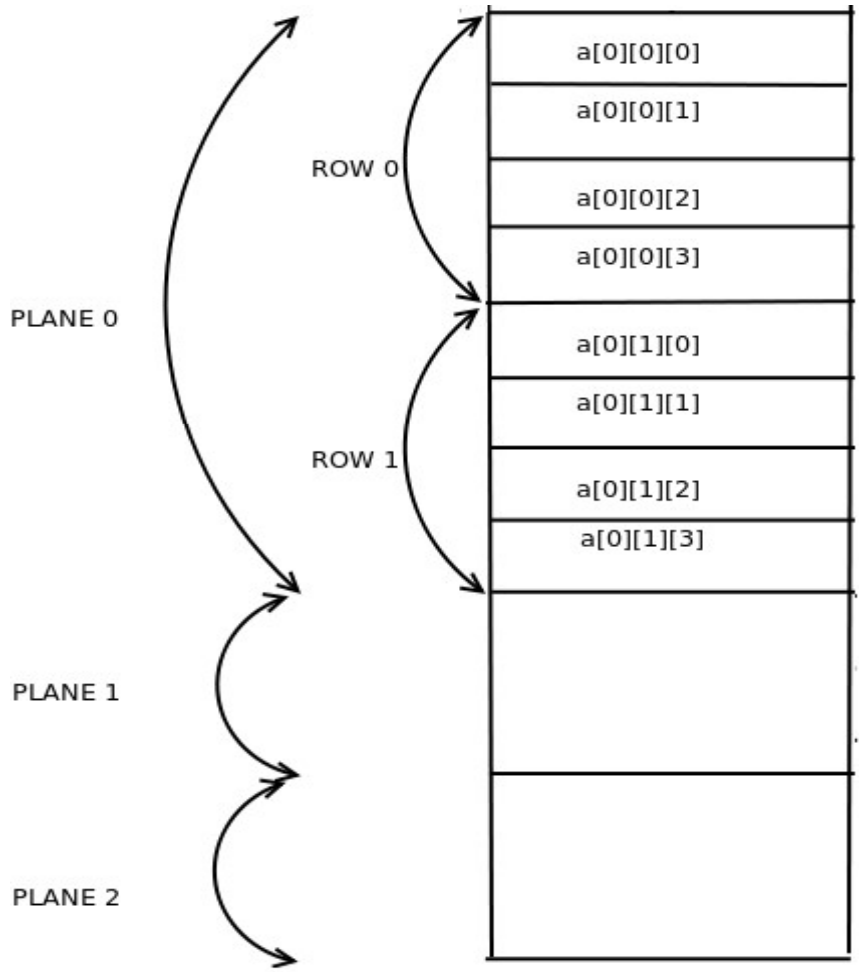
= 1650

3-Dimensional Array

- A 3-dimensional array is modeled as a collection of identical 2-dimensional arrays placed one behind the other or one above.
- Three subscript are needed to access a particular element.
- First subscript specifies the plane, second subscript the row and the third subscript the column.

3-Dimensional Array

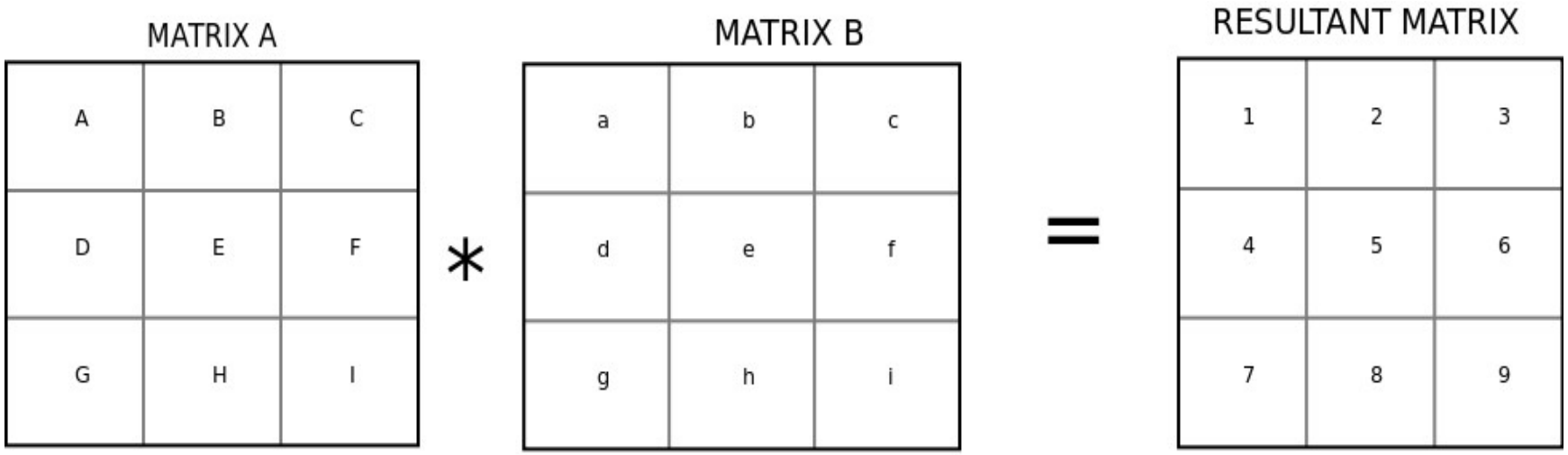
Representing a
3-dimensional array



Matrix Multiplication

- Given two matrices A(m*n) and B(x*y).
- Condition for multiplication is that n==x.
- If n != x, matrix multiplication is not possible.

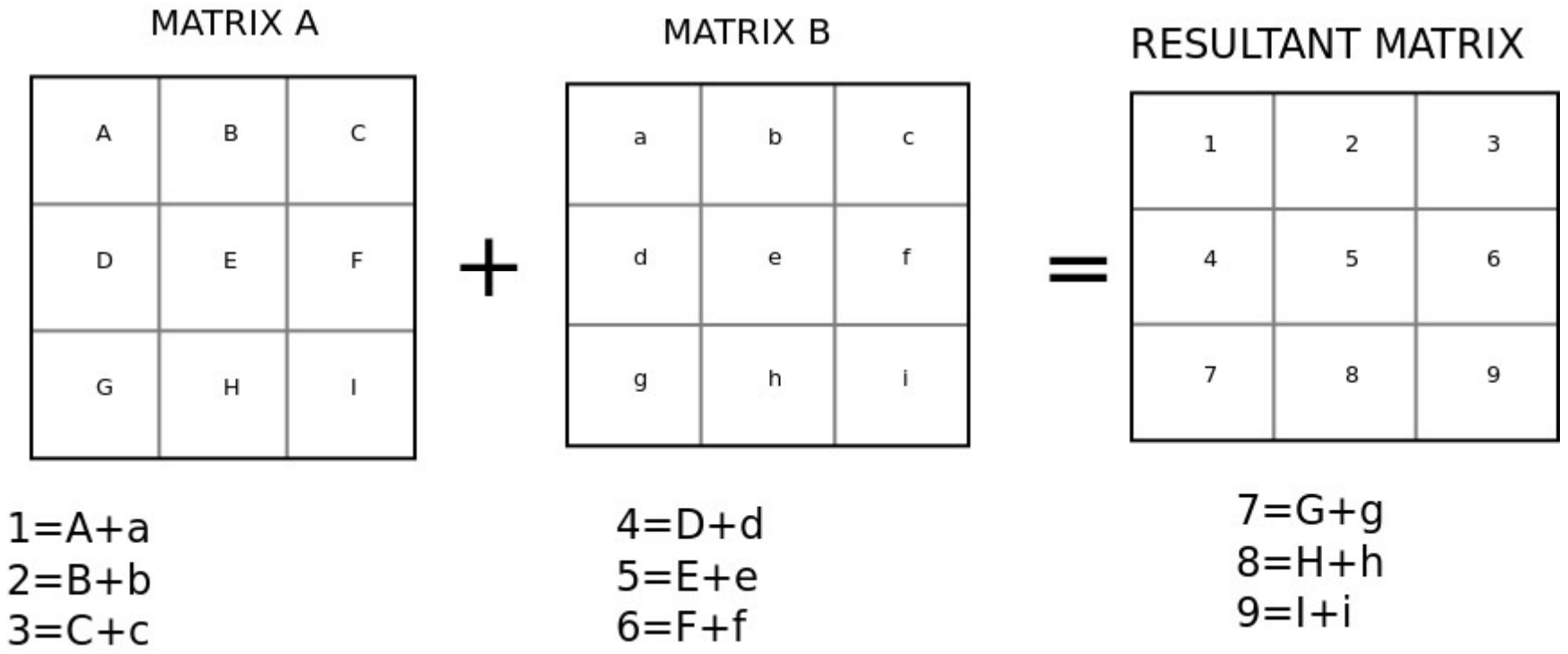
Matrix Multiplication



1=A.a+B.d+C.g
2=A.b+B.e+C.h
3=A.c+B.f+C.i
4=D.a+E.d+F.g
5=D.b+E.e+F.h
6=D.c+E.f+F.i
7=G.a+H.d+I.g
8=G.b+H.e+I.h
9=G.c+H.f+I.i

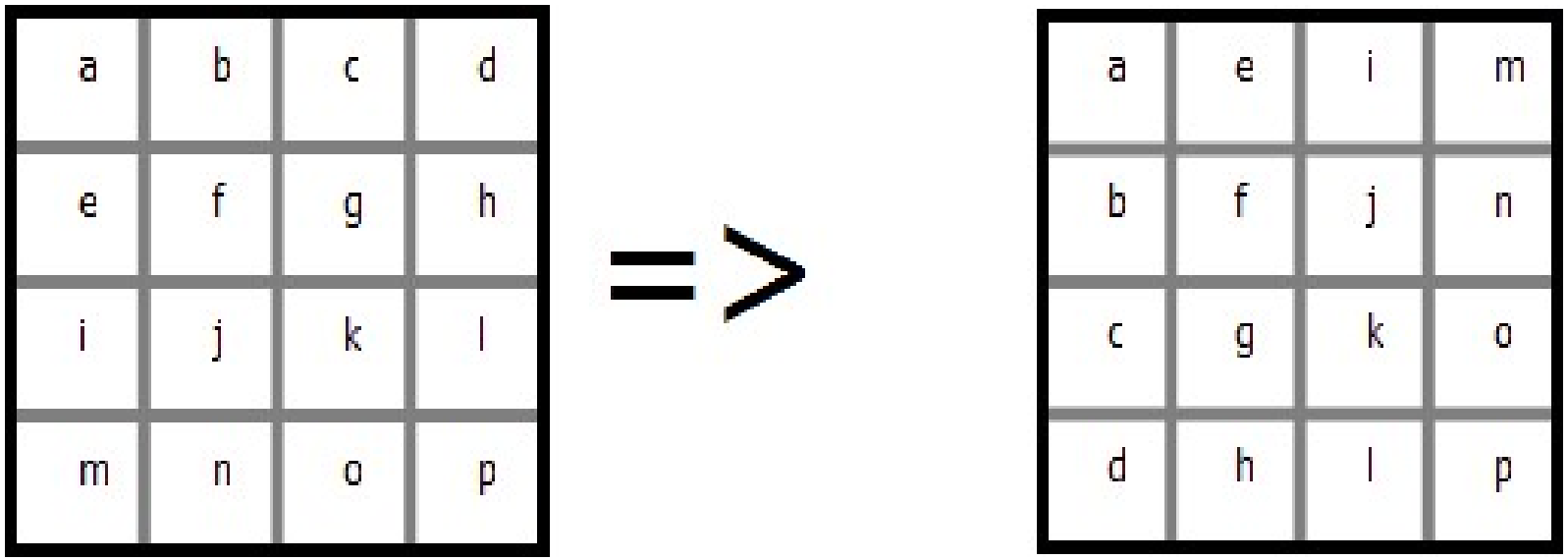
Data Structures: A Programming Approach with C, PHI,
Dharmender Singh Kushwaha & A.K.Misra

Matrix Addition



Data Structures: A Programming Approach with C, PHI,
Dharmender Singh Kushwaha & A.K.Misra

Matrix Transpose



Pointer & 1-D Array

```
int array [s], *ip;          //pointer to int
ip= & array[3]               //pointer points element of array
main()
{
    int n,*p;                // defines two variables
    p=&n;                    // pointer p has address of or points to variable n
    *p=0;                    // set n=0
    *p + =1;                 // increment what p point to
    (*p)+ +;                 // ----,,----
}
```

Array of Pointers

- Suppose we have a declaration:
  - int *a[10] // array of pointers which are 10 in number.
- Let there be other array marks declared as:
  - int marks[10]={1,2,3,4,5,6,7,8,9,0};
- These 10 elements of the array marks will be stored in certain addresses.
- Now these add. can be stored in array of pntrs.
- Address of element 1 is the first element of int *a[] i.e. the first element of int marks[10] and so on. This is array of pointer.

Pointer to an Array

- Consider the declaration:
  - int(*a)[10] // here ‘a’ is a pointer to an array containing 10 integers.
- Now if we had the following lines of code:
  - int marks[10];
  - a=&marks[10];
- Here, we deference pointer a with '*'operator and we first get element of int marks[10] array that is marks[0] and so on.
- This is pointer to array.

Array within Structures: Complex Structure

- Structure can contains arrays or other structures so it is sometimes called complex structure.

```
struct address
{
    int house_no;
    char street_name [40];
    char city_name [30];
    int pin_code;
};
```

Array of Structure

- Once we need to store the complex structures, the natural choice is array of structures.

```
Struct student
{
    char name[12];
    int rollno;
    int percent_marks;
};
student studresult[60];
```

Summary

- Array is a data structure consisting of a collection of elements, each identified by at least one index.
- A one-dimensional is a type of linear array.
- Accessing its elements involves a single subscript which can either represent a row or column index.

Summary

- Static arrays have a size that is fixed at allocation time and consequently do not allow elements to be inserted or removed.

- Array stores values in contiguous memory location.

- Arrays take linear (O (*n*)) space in the number of elements *n* that they hold.