

Data Structures

Dr. Balu L. Parne

Introduction to Data Structures

Introduction

- Data structure is a storage that is used to store and organize data. It is a way of arranging data on a computer so that it can be accessed and updated efficiently.
- Depending on your requirement and project, it is important to choose the right data structure for your project. For example, if you want to store data sequentially in the memory, then you can go for the Array data structure.

Introduction

- Data structure is representation of the logical relationship existing between individual elements of data.
- In other words, a data structure is a way of organizing all data items that considers not only the elements stored but also their relationship to each other.
- Data structure affects the design of both structural & functional aspects of a program.

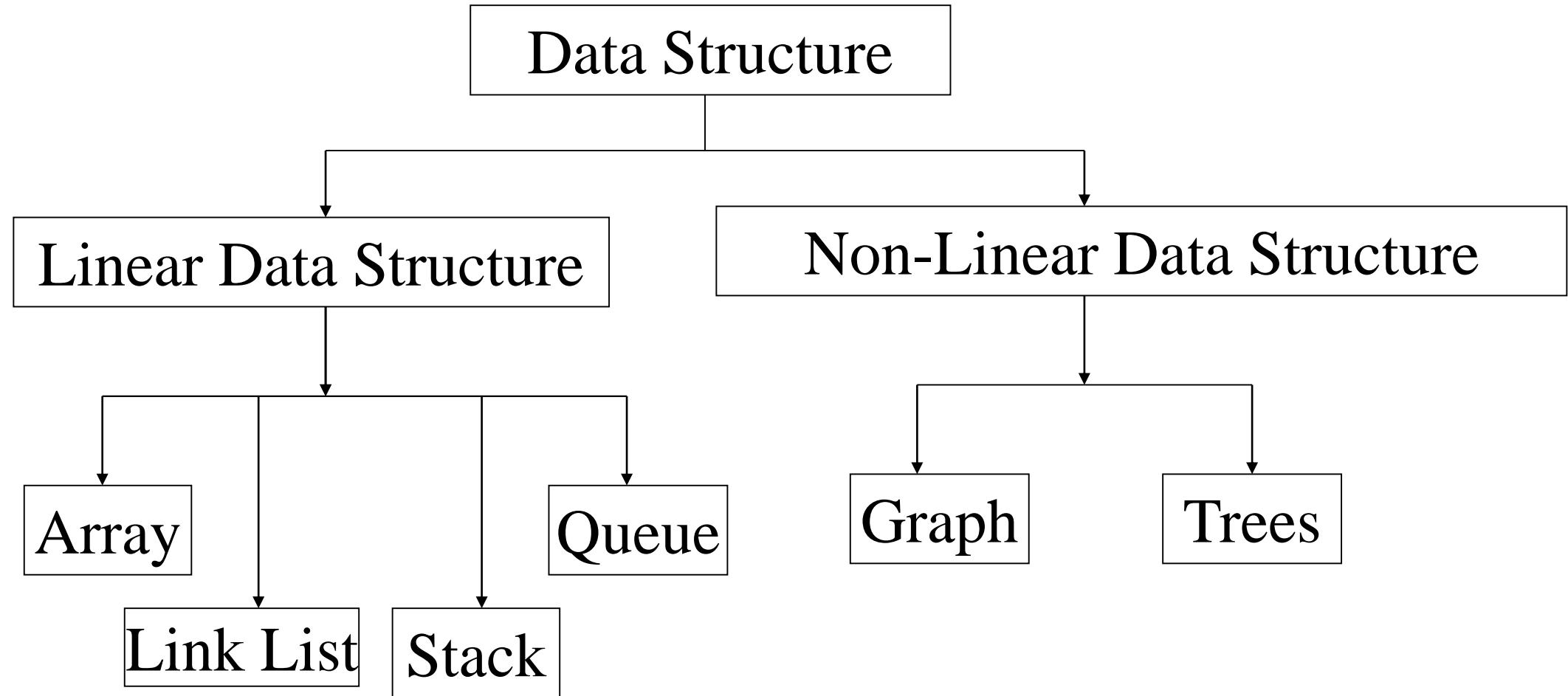
Program=algorithm + Data Structure

- You know that a algorithm is a step by step procedure to solve a particular function.

Introduction

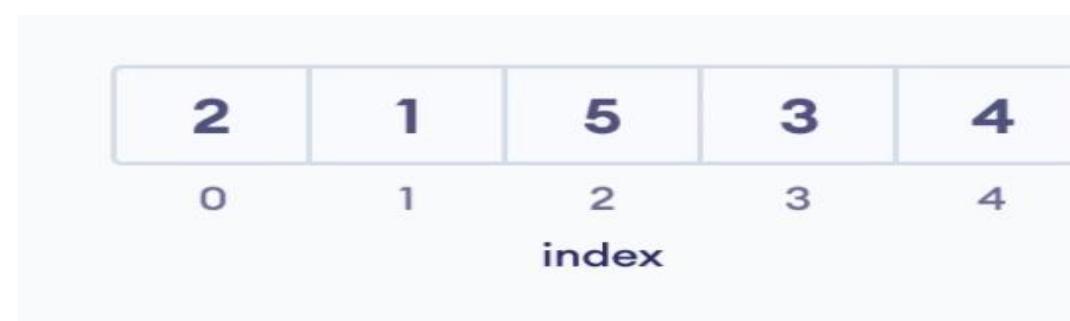
- That means, algorithm is a set of instruction written to carry out certain tasks & the data structure is the way of organizing the data with their logical relationship retained.
- To develop a program of an algorithm, we should select an appropriate data structure for that algorithm.
- Therefore algorithm and its associated data structures from a program.

Classification of Data Structure



Array Data Structure :

- In an array, elements in memory are arranged in continuous memory. All the elements of an array are of the same type. And, the type of elements that can be stored in the form of arrays is determined by the programming language.



Linked Lists Data Structure:

- In linked list data structure, data elements are connected through a series of nodes. And, each node contains the data items and address to the next node.

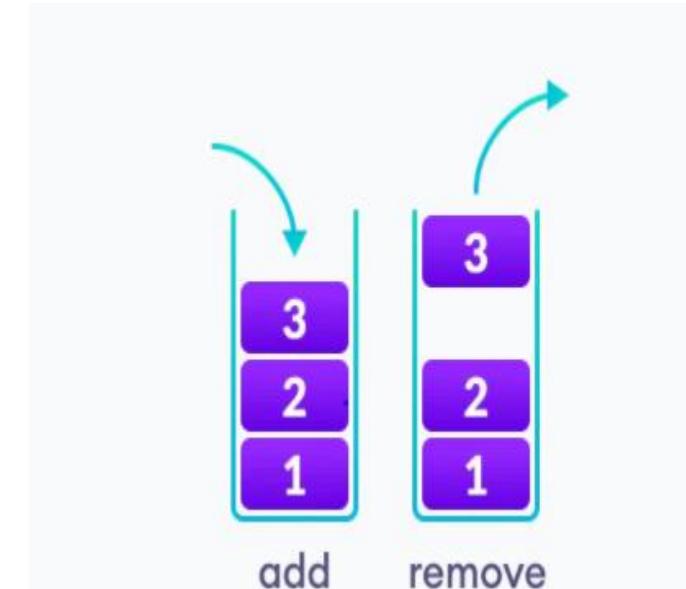
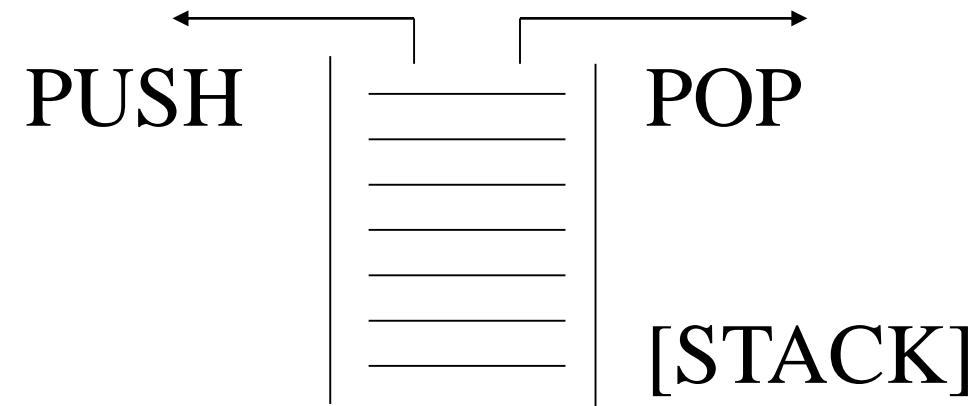


Stack Data Structure:

- A stack is also an ordered collection of elements like arrays, but it has a special feature that deletion and insertion of elements can be done only from one end called the top of the stack (TOP).
- Due to this property it is also called as last in first out type of data structure (LIFO).
- It could be through of just like a stack of plates placed on table in a party, a guest always takes off a fresh plate from the top and the new plates are placed on to the stack at the top.
- When an element is inserted into a stack or removed from the stack, its base remains fixed where the top of stack changes.

Stack Data Structure:

- Insertion of element into stack is called PUSH and deletion of element from stack is called POP.
- The below figure show how the operations take place on a stack:

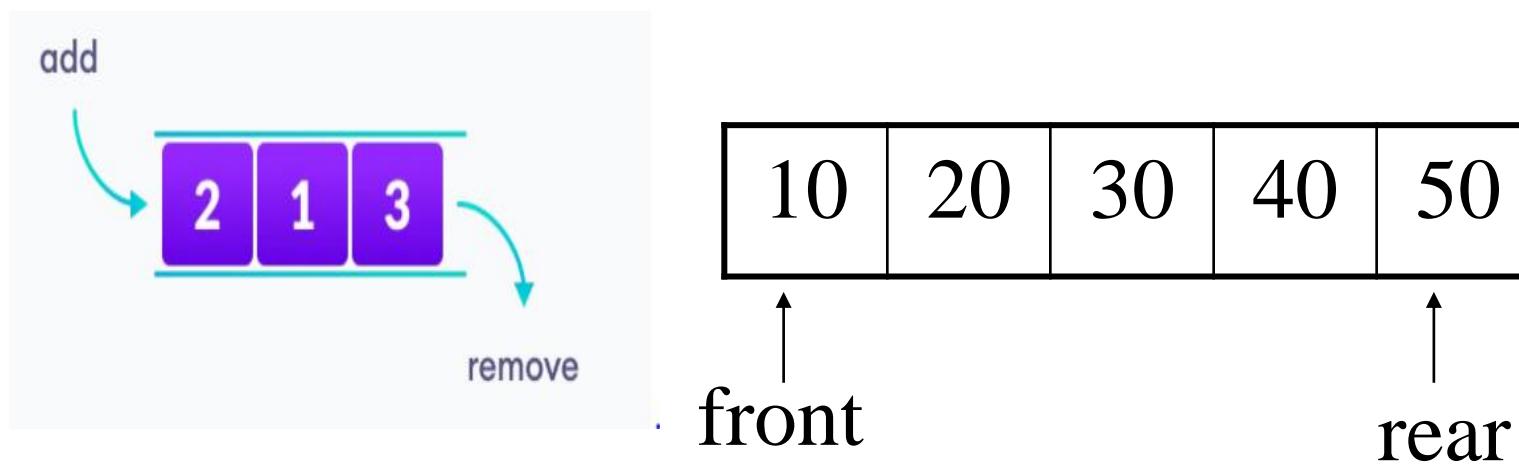


Queue Data Structure:

- Queue are first in first out type of data structure (i.e. FIFO)
- In a queue new elements are added to the queue from one end called REAR end and the element are always removed from other end called the FRONT end.
- The people standing in a railway reservation row are an example of queue.

Queue Data Structure:

- Each new person comes and stands at the end of the row and person getting their reservation confirmed get out of the row from the front end.
- The below figure shows how the operations take place on a queue:

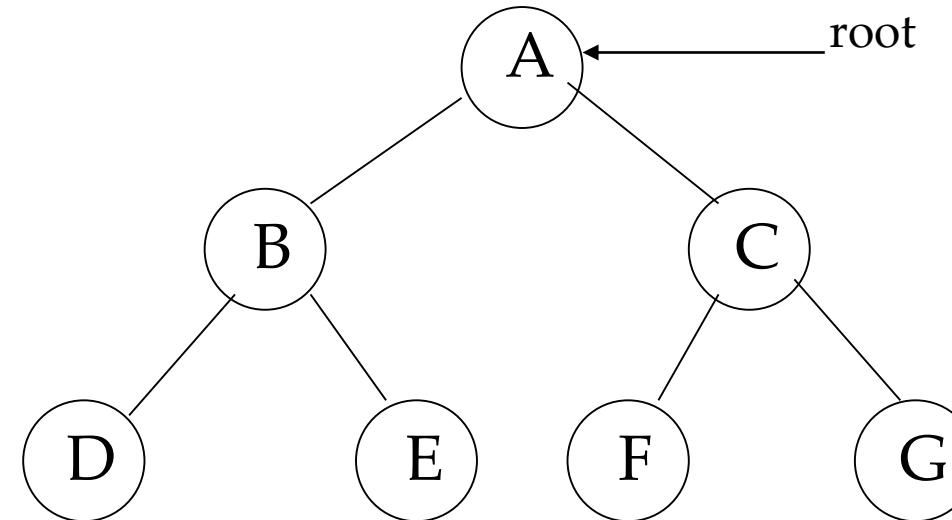


Tree Data Structure:

- A tree can be defined as finite set of data items (nodes).
- Tree is non-linear type of data structure in which data items are arranged or stored in a sorted sequence.
- Tree represent the hierarchical relationship between various elements.
- There is a special data item at the top of hierarchy called the Root of the tree.
- The remaining data items are partitioned into number of mutually exclusive subset, each of which is itself, a tree which is called the sub tree.
- The tree always grows in length towards bottom in data structures, unlike natural trees which grows upwards.

Tree Data Structure:

- The tree structure organizes the data into branches, which related the information.

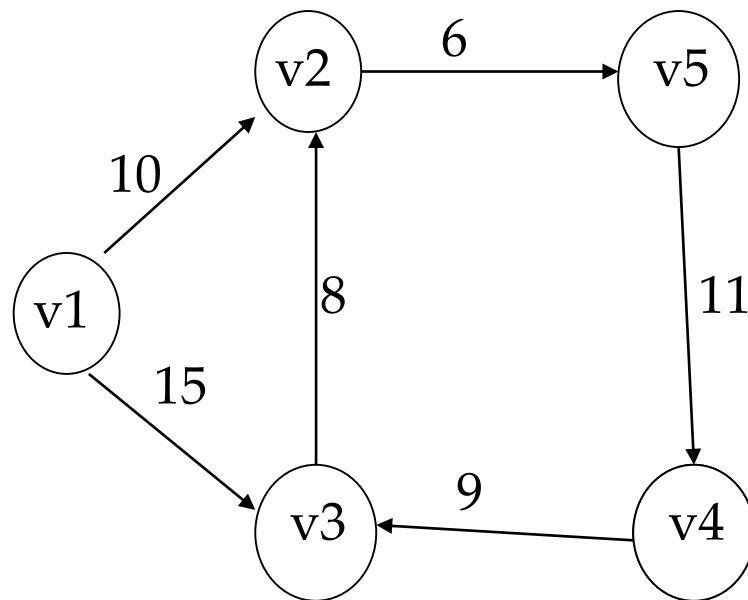


Graph Data Structure:

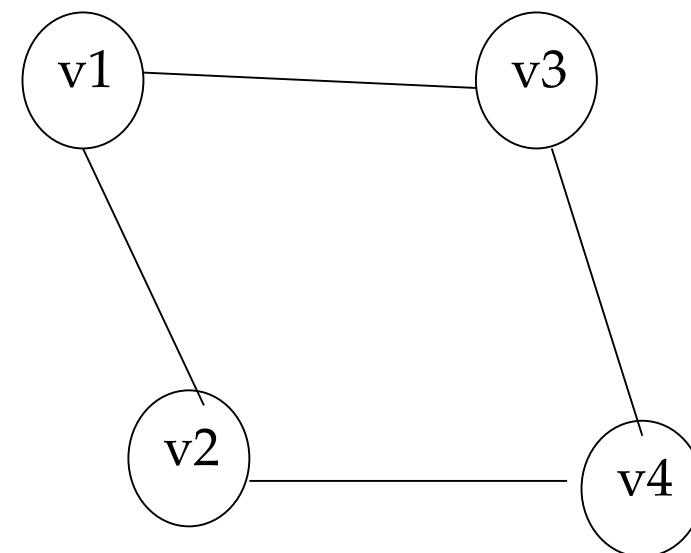
- Graph is a mathematical non-linear data structure capable of representing many kind of physical structures.
- It has found application in Geography, Chemistry and Engineering sciences.
- Definition: A graph $G(V,E)$ is a set of vertices V and a set of edges E .
- An edge connects a pair of vertices and many have weight such as length, cost and another measuring instrument for according the graph.
- Vertices on the graph are shown as point or circles and edges are drawn as arcs or line segment.

Graph Data Structure:

- Example of graph:



[a] Directed & Weighted Graph



[b] Undirected Graph

Linear Data Structures	Non Linear Data Structures
The data items are arranged in sequential order, one after the other.	The data items are arranged in non-sequential order (hierarchical manner).
All the items are present on the single layer.	The data items are present at different layers.
It can be traversed on a single run. That is, if we start from the first element, we can traverse all the elements sequentially in a single pass.	It requires multiple runs. That is, if we start from the first element it might not be possible to traverse all the elements in a single pass.
The memory utilization is not efficient.	Different structures utilize memory in different efficient ways depending on the need.
The time complexity increase with the data size.	Time complexity remains the same.
Example: Arrays, Stack, Queue	Example: Tree, Graph, Map

What is Data Types:

WHAT IS A DATA TYPE?

Two important things about data types:

1. Defines a certain domain of values.
2. Defines Operations allowed on those values.

EXAMPLE:

```
int type
- Takes only integer values
- Operations: addition, subtraction,
multiplication, bitwise operations etc.
```

User Defined Data Types:

USER DEFINED DATA TYPES

In contrast to primitive data types, there is a concept of user defined data types.

The operations and values of user defined data types are not specified in the language itself but is specified by the user.

EXAMPLE: Structure, union and enumeration.

By using structures, we are defining our own type by combining other data types.

```
struct point {  
    int x;  
    int y;  
};
```

Abstract Data Types

ABSTRACT DATA TYPES (ADT)

ADTs are like user defined data types which defines operations on values using functions without specifying what is there inside the function and how the operations are performed.

EXAMPLE: Stack ADT

A stack consists of elements of same type arranged in a sequential order.

Operations:

 initialize() - initializing it to be empty

Push() - Insert an element into the stack

Pop() - Delete an element from the stack

isEmpty() - checks if stack is empty

isFull() - checks if stack is full

ONLY BLUEPRINT

Abstract Data Types

ABSTRACT DATA TYPES (ADT)

Think of ADT as a **black box** which hides the inner structure and design of the data type from the user.

There are multiple ways to implement an ADT.

EXAMPLE:

A stack ADT can be implemented using arrays or linked lists.



Why ADT

WHY ADT?

The program which uses data structure is called a **client** program
It has access to the ADT i.e. interface.

The program which implements the data structure is known as the **implementation**.

ADVANTAGE

Let say, if someone wants to use the stack in the program, then he can simply use push and pop operations without knowing its implementation.

Also, if in future, the implementation of stack is changed from array to linked list, then the client program will work in the same way without being affected.

Arrays

Why we need Array?

- What will be output of the following program??

```
main( )  
{  
    int x ;  
    x = 5 ;  
    x = 10 ;  
    printf ( "\nx = %d", x ) ;  
}
```

No doubt, this program will print the value of x as 10. Why so? Because when a value 10 is assigned to x, the earlier value of x, i.e. 5, is lost. Thus, ordinary variables (the ones which we have used so far) are capable of holding only one value at a time (as in the above example). However, there are situations in which we would want to store more than one value at a time in a single variable.

- For example, suppose we wish to arrange the percentage marks obtained by 100 students in ascending order. In such a case we have two options to store these marks in memory:
 - Construct 100 variables to store percentage marks obtained by 100 different students, i.e. each variable containing one student's marks.
 - Construct one variable (called array or subscripted variable) capable of storing or holding all the hundred values.
- Which is the Best Option??
- Obviously, the second alternative is better. A simple reason for this is, it would be much easier to handle one variable than handling 100 different variables.

Arrays - Introduction

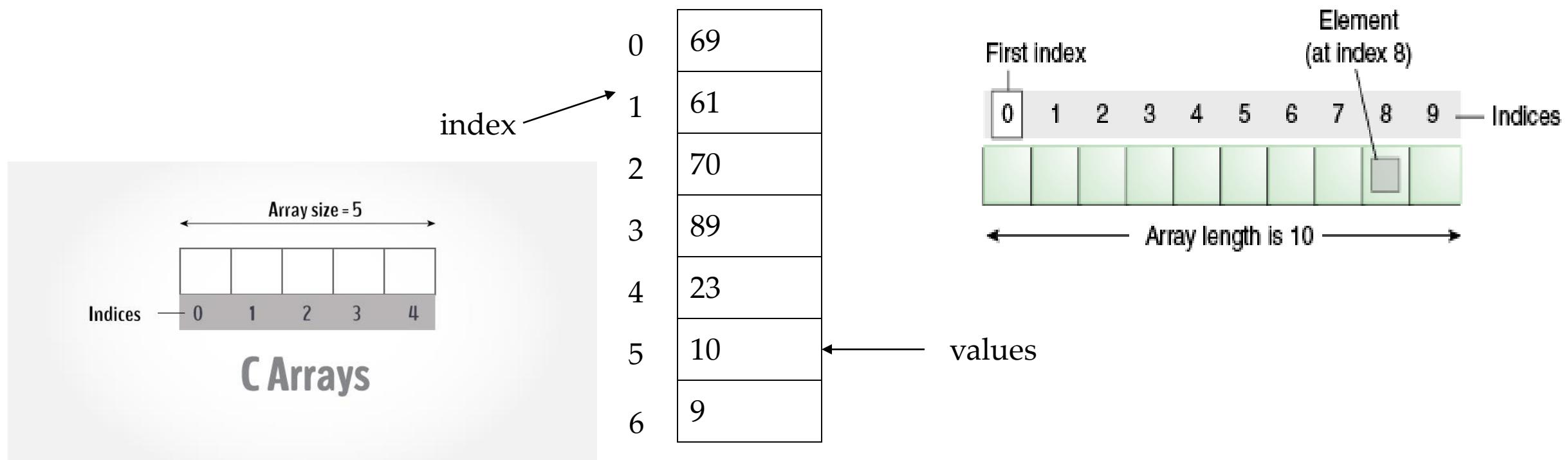
- An array is a collection of similar type of elements that have a contiguous memory location.
- An array is a group of contiguous or related data items that share a common name.
- An array is a collective name given to a group of 'similar quantities'. These similar quantities could be percentage marks of 100 students, or salaries of 300 employees, or ages of 50 employees.
- Used when programs have to handle large amount of data.
- Each value is stored at a specific position.
- Position is called a index or subscript. Base index = 0

- For example, assume the following group of numbers, which represent percentage marks obtained by five students.

per = { 48, 88, 34, 23, 96 }

If we want to refer to the second number of the group, the usual notation used is **per₂**. Similarly, the fourth number of the group is referred as **per₄**. However, in C, the fourth number is referred as **per[3]**. This is because in C the counting of elements begins with 0 and not with 1. Thus, in this example **per[3]** refers to 23 and **per[4]** refers to 96. In general, the notation would be **per[i]**, where, **i** can take a value 0, 1, 2, 3, or 4, depending on the position of the element being referred. Here **per** is the subscripted variable (array), whereas **i** is its subscript.

Arrays - Introduction



Note: Any given array must be of the same type. i.e. we cannot have an array of 10 numbers, of which 5 are **ints** and 5 are **floats**.

How to declare an Array?

- An array needs to be declared so that the compiler will know what kind of an array and how large an array we want.

- The syntax for array declaration is:

```
dataType arrayName[arraySize];
```

- An array is a variable that can store multiple values. For example, if you want to store 100 integers, you can create an array for it.

```
int data[100];
```

- we declared an array, mark, of floating-point type. And its size is 5.

```
float mark[5];
```

- **Note: the size and type of an array cannot be changed once it is declared.**

How to declare an Array?

- For Example

```
int marks[30];
```

Here, **int** specifies the type of the variable, just as it does with ordinary variables and the word **marks** specifies the name of the variable. The **[30]** however is new. The number 30 tells how many elements of the type **int** will be in our array. This number is often called the ‘dimension’ of the array. The bracket ([]) tells the compiler that we are dealing with an array.

```
int num[35]; /* An integer array of 35 elements */  
char ch[10]; /* An array of characters for 10 elements */
```

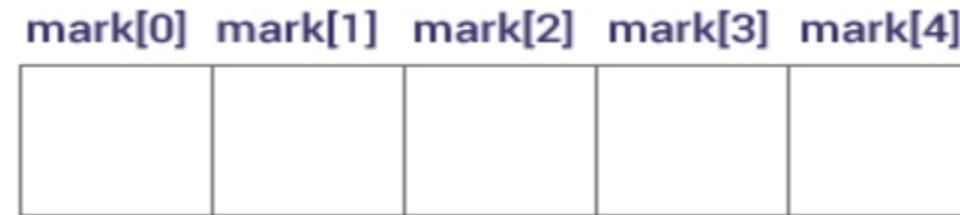
Accessing Array Elements:

- An array subscript (or index) can be used to access any element stored in a array. Subscript starts with 0, which means **arr[0]** represents the first element in the array arr.
- In general **arr[n-1]** can be used to access **nth** element of an array. where n is any integer number.
- For example: We have declare integer array mydata with 20 elements in it as follow:

```
int mydata[20];  
mydata[0] /* first element of array mydata*/  
mydata[19] /* last (20th) element of array mydata*/
```

Accessing Array Elements:

- Suppose you declared an array **mark**. The first element is **mark[0]**, the second element is **mark[1]** and so on.



- Arrays have 0 as the first index, not 1. In this example, **mark[0]** is the first element.
- If the **size** of an array is **n**, to access the last element, the **n-1 index** is used. In this example, **mark[4]**.

Initialization of an Array:

- We can initialize the array during declaration as follow:

```
int mark[5] = {19, 10, 8, 17, 9};
```

- We can also initialize an array as follow:

```
int mark[] = {19, 10, 8, 17, 9};
```

- Here, the size of an array is not specified. However, the compiler knows its size is 5 as we are initializing it with 5 elements.

mark[0] mark[1] mark[2] mark[3] mark[4]

19	10	8	17	9
----	----	---	----	---

Initialization of one-d array...

Compile time initialization:

```
type array-name[size] = {list of values};
```

```
int number[3]={0,0,0};
```

- Declare **number** as an array of size 3 and initialize each element to 0

```
float total[5]={0.0,15.3,6.2};
```

- Declare **total** as an array of size 5 and initialize first three elements to 0.0, 15.3 and 6.2 and remaining elements to 0.0

```
int number[]={0,0,0};
```

- Declare **number** as an array of size 3 and initialize each element to 0

Initialization of one-d array...

```
char name[6]={ 'h' , 'e' , 'l' , 'l' , 'o' , '\0' };
```

- Declare name as a character array of size 6 and initialize it to “hello”.
- Same as,

```
char name[6] = "hello";
```

- The following is illegal in C

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

More elements than array size !!!

Initialization of one-d array...

- Run time initialization

```
int x[10];           // Declaration of array
for(i=0;i<10;i++)
{
    printf("Enter a number\n");
    scanf("%d",&x[i]); // Initialization of array elements
}
```

Input Array Elements:

- We can take input from the user and store it in an array element as follow:

```
// take input and store it in the 3rd element  
scanf("%d", &mark[2]);
```

```
// take input and store it in the ith element  
scanf("%d", &mark[i-1]);
```

Output Array Elements:

- We can print an individual element of an array as follow:

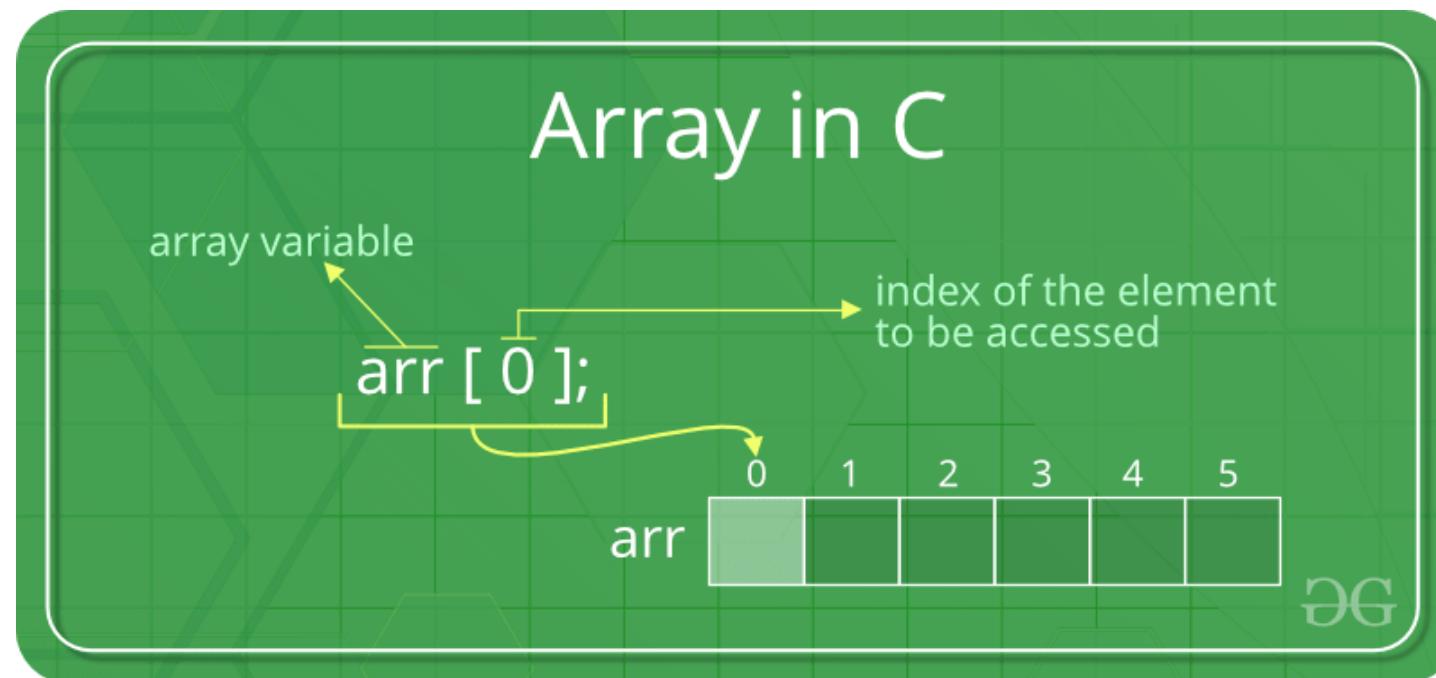
```
// print the first element of the array  
printf("%d", mark[0]);
```

```
// print the third element of the array  
printf("%d", mark[2]);
```

```
// print ith element of the array  
printf("%d", mark[i-1]);
```

Accessing Array Elements:

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



Access elements out of its bound!

- Suppose you declared an array of 10 elements. Let's say,

```
int testArray[10];
```

- We can access the array elements from **testArray[0]** to **testArray[9]**.
- Now let's say if you try to access **testArray[12]**.
- **The element is not available. This may cause unexpected output (undefined behavior). Sometimes you might get an error and some other time your program may run correctly.**
- Hence, we should never access elements of an array outside of its bound.

Access elements out of its bound!

What will be the output of the following Program??

```
#include <stdio.h>

int main()
{
    int arr[2];
    printf("%d \n", arr[3]);
    printf("%d \n", arr[-2]);

    return 0;
}
```

In this program, we are trying to access the out of bound values of an array as we have define array size is 2 and trying to access values at index 3 and -2 respectively. So, compiler will print some unexpected value as output as shown in output window.



Access elements out of its bound!

```
#include <stdio.h>
int main()
{
    // Array declaration by initializing it with more
    // elements than specified size.
    int arr[2] = { 10, 20, 30, 40, 50 };
    | return 0;
}
```

In C, it is not compiler error to initialize an array with more elements than the specified size. For example, the below program compiles fine and shows just Warning.

Line	Col	File	Message
		C:\Users\balup\Desktop\C Programs\Arrays\Initialize...	In function 'main':
6	5	C:\Users\balup\Desktop\C Programs\Arrays\InitializeArr...	[Warning] excess elements in array initializer
6	5	C:\Users\balup\Desktop\C Programs\Arrays\InitializeArr...	[Warning] (near initialization for 'arr')
6	5	C:\Users\balup\Desktop\C Programs\Arrays\InitializeArr...	[Warning] excess elements in array initializer
6	5	C:\Users\balup\Desktop\C Programs\Arrays\InitializeArr...	[Warning] (near initialization for 'arr')

- Write a Program to take 5 values from the user and store them in an array and Print the elements stored in the array.

```
#include <stdio.h>
int main()
{
    int values[5];
    int i;
    printf("Enter value of an array elements: \n");
    // taking input and storing it in an array
    for(i = 0; i < 5; i++)
    {
        scanf("%d", &values[i]);
    }
    printf("Display value of an array elements: \n");
    // printing elements of an array
    for(i = 0; i < 5; i++)
    {
        printf("%d\n", values[i]);
    }
    return 0;
}
```

- Write a program to print sum and average of 10 values stored in an arrays of type integer.

```
#include<stdio.h>
int main()
{
    int numbers [] = {2, -9, 0, 5, 12, -25, 22, 9, 8, 12};
    int i, sum = 0;
    float average;

    for (i=0; i<10; i++)
    {
        sum = sum+numbers[i];
    }

    average = sum / 10;

    printf("Sum = %d\n", sum);
    printf("Average = %f\n", average);
    return 0;
}
```

```
#include<stdio.h>
int main()
{
    int numbers [10]; // Array size is fixed everytime we execute code.
    int i, sum = 0;
    float average;
    printf("Enter the elements of an array:");
    for (i=0; i<10; i++)
    {
        scanf("%d", &numbers[i]); // To take input from the user.
    }
    for(i=0; i<10; i++)
    {
        sum = sum+numbers[i]; // Compute sum of all the elements.
    }

    average = sum / 10;
    printf("Sum = %d\n", sum);
    printf("Average = %f\n", average);
    return 0;
}
```

```
int main()
{
    int i, size, sum = 0;
    float average;
    printf("Enter the size of an array");
    scanf("%d", &size);
    int numbers [size]; // Array size is user define.
    printf("Enter the elements of an array:");
    for (i=0; i<size; i++)
    {
        scanf("%d", &numbers[i]); // To take input from the user.
    }
    for(i=0; i<size; i++)
    {
        sum = sum+numbers[i]; // Compute sum of all the elements.
    }
    average = sum / 10;
    printf("Sum = %d\n", sum);
    printf("Average = %f\n", average);
    return 0;
}
```

Summary:

- An array is a collection of similar elements.
- The first index in the array is numbered 0, so the last element is 1 less than the size of the array.
- An array is also known as a subscripted variable.
- Before using an array its type and dimension must be declared.
- However big an array its elements are always stored in contiguous memory locations.

Home Work Problem:

- Write a program to create an array of 10 numbers and find the largest number among them.
- Write a program to create an array of 10 numbers and find the smallest number among them.
- Write a program to search whether the given element is present in an array or not using linear search.
- Write a program to find maximum and minimum elements in 1-D integer array.
- Write a program to merge elements of two different arrays into third array.

Two Dimensional Arrays

Two dimensional arrays :

- When we need to store a table of values
- Consider the data structure that shows the value of sales of three items by four sales girls

	Item 1	Item 2	Item 3
Sales girl 1	310	234	110
Sales girl 2	112	345	321
Sales girl 3	102	321	213
Sales girl 4	250	321	321

- C allows us to define such table by using two dimensional array

Two dimensional arrays :

- It is also possible for arrays to have two or more dimensions. The two-dimensional array is also called a matrix.
- C programming language allows multidimensional arrays. Here is the general form of a multidimensional array declaration:

type Arrayname [size1][size2].....[sizeN];

- For example, the following declaration creates a three dimensional integer array –

int threedim[5][10][4];

Two dimensional arrays :

- The simplest form of multidimensional array is the two-dimensional array.
- A two-dimensional array is, in essence, a list of one-dimensional arrays.
- To declare a two-dimensional integer array of size [x][y], you would write something as follows –

type arrayName [x][y];

- Where **type** can be any valid C data type and **arrayName** will be a valid C identifier. A two-dimensional array can be considered as a table which will have x number of rows and y number of columns.

Two dimensional arrays...

- Declaration

type array-name[row_size][col_size];

- Representation of 2D array in memory

	Col 0	Col 1	Col 2
Row 0	[0][0] 310	[0][1] 275	[0][2] 365
Row 1	[1][0] 10	[1][1] 190	[1][2] 325
Row 2	[2][0] 405	[2][1] 235	[2][2] 240
Row 3	[3][0] 310	[3][1] 275	[3][2] 365

Two dimensional arrays :

- We can declare a two dimensional array X of float type as follow:

float x[3][4]

- Here, x is a two-dimensional (2d) array. The array can hold 12 elements. You can think the array as a table with 3 rows and each row has 4 columns.

	Column 1	Column 2	Column 3	Column 4
Row 1	x[0][0]	x[0][1]	x[0][2]	x[0][3]
Row 2	x[1][0]	x[1][1]	x[1][2]	x[1][3]
Row 3	x[2][0]	x[2][1]	x[2][2]	x[2][3]

Initialization of Two dimensional arrays :

- Multidimensional arrays may be initialized by specifying bracketed values for each row.
- Following is an array with 3 rows and each row has 4 columns.

```
int a[3][4] = {  
    {0, 1, 2, 3} , /* initializers for row indexed by 0 */  
    {4, 5, 6, 7} , /* initializers for row indexed by 1 */  
    {8, 9, 10, 11} /* initializers for row indexed by 2 */  
};
```

- The nested braces, which indicate the intended row, are optional. The following initialization is equivalent to the previous example –

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

Initialization of Two dimensional arrays :

- In C programming, initialization of a multidimensional arrays **can have left most dimension as optional. Except the left most dimension, all other dimensions must be specified.**
- **Following are compiled time initialization of 2d array:**

```
// Different ways to initialize two-dimensional array
```

```
int c[2][3] = {{1, 3, 0}, {-1, 5, 9}};  
  
int c[][3] = {{1, 3, 0}, {-1, 5, 9}};  
  
int c[2][3] = {1, 3, 0, -1, 5, 9};
```

```
/* Valid declaration*/  
int abc[2][2] = {1, 2, 3 ,4 }  
/* Valid declaration*/  
int abc[][2] = {1, 2, 3 ,4 }  
/* Invalid declaration - you must specify second dimension*/  
int abc[][] = {1, 2, 3 ,4 }  
/* Invalid because of the same reason mentioned above*/  
int abc[2][] = {1, 2, 3 ,4 }
```

Initialization of Two dimensional arrays :

What will be output of the following program??

```
#include<stdio.h>
int main()
{
    int a[][][2] = { {{1, 2}, {3, 4}},
                    {{5, 6}, {7, 8}}
                    };
    printf("%d", sizeof(a));
    getchar();
    return 0;
}
```

This program gives an error as all right most dimension are not specified.

```
/* Valid declaration*/
int abc[2][2] = {1, 2, 3 ,4 }

/* Valid declaration*/
int abc[][2] = {1, 2, 3 ,4 }

/* Invalid declaration - you must specify second dimension*/
int abc[][] = {1, 2, 3 ,4 }

/* Invalid because of the same reason mentioned above*/
int abc[2][] = {1, 2, 3 ,4 }
```

Initialization of Two dimensional arrays :

- Run-time initialization

```
int sales[4][3];
for(int i=0;i<4;i++)
{
    for(int j=0;j<3;j++)
    {
        scanf("%d", &sales[i][j]);
    }
}
```

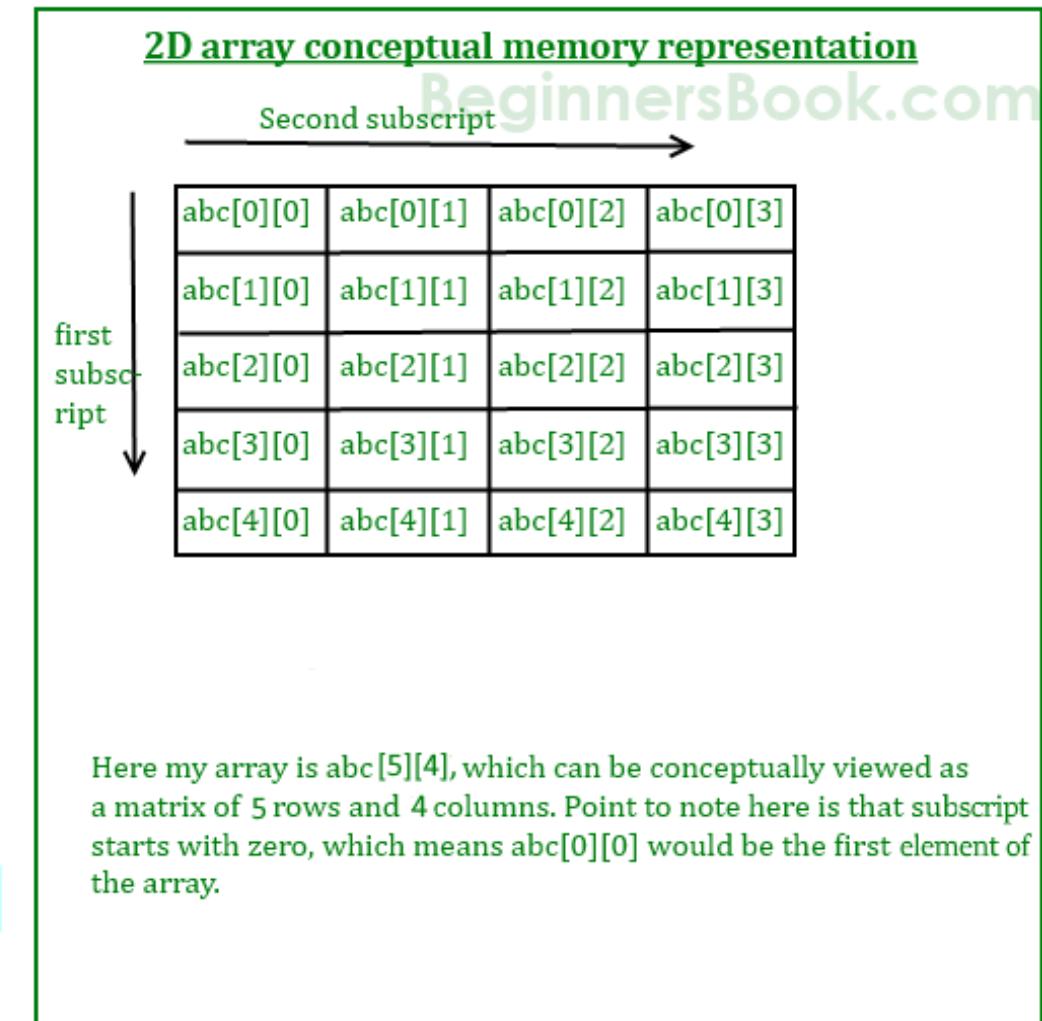
This is the example of runtime initialization of an 2D Array. Here we have define integer type of array sales with 4 rows and 3 columns. User will enter the 12 values that will initialize all the elements in the array.

How to store user input data into 2D array

- The given 2D array arr[n1][n2] can have $n1 \times n2$ elements. So the array abc[5][4] can have $5 \times 4 = 20$ elements.
- To store the elements entered by user we are using two for loops, one of them is a nested loop.
- The outer loop runs from 0 to the (rowsize -1) and the inner for loops runs from 0 to the (colsiz -1). This way the order in which user enters the elements would be abc[0][0], abc[0][1], abc[0][2]...so on.
- The following program will illustrate in details:

How to store user input data into 2D array

```
int main()
{
    int abc[5][4]; /* 2D array declaration*/
    int i, j; /*Counter variables for the loop*/
    for(i=0; i<5; i++)
    {
        for(j=0;j<4;j++)
        {
            printf("Enter value for abc[%d][%d]:", i, j);
            scanf("%d", &abc[i][j]);
        }
    }
    return 0;
}
```



How to store user input data into 2D array

However the actual representation of this array in memory would be something like this:



memory locations for the array elements

Array is of integer type so each element would use 4 bytes that's the reason there is a difference of 4 in element's addresses.

The addresses are generally represented in hex. This diagram shows them in integer just to show you that the elements are stored in contiguous locations, so that you can understand that the address difference between each element is equal to the size of one element(int size 4). For better understanding see the program below.

Actual memory representation of a 2D array

Accessing Two-Dimensional Array Elements:

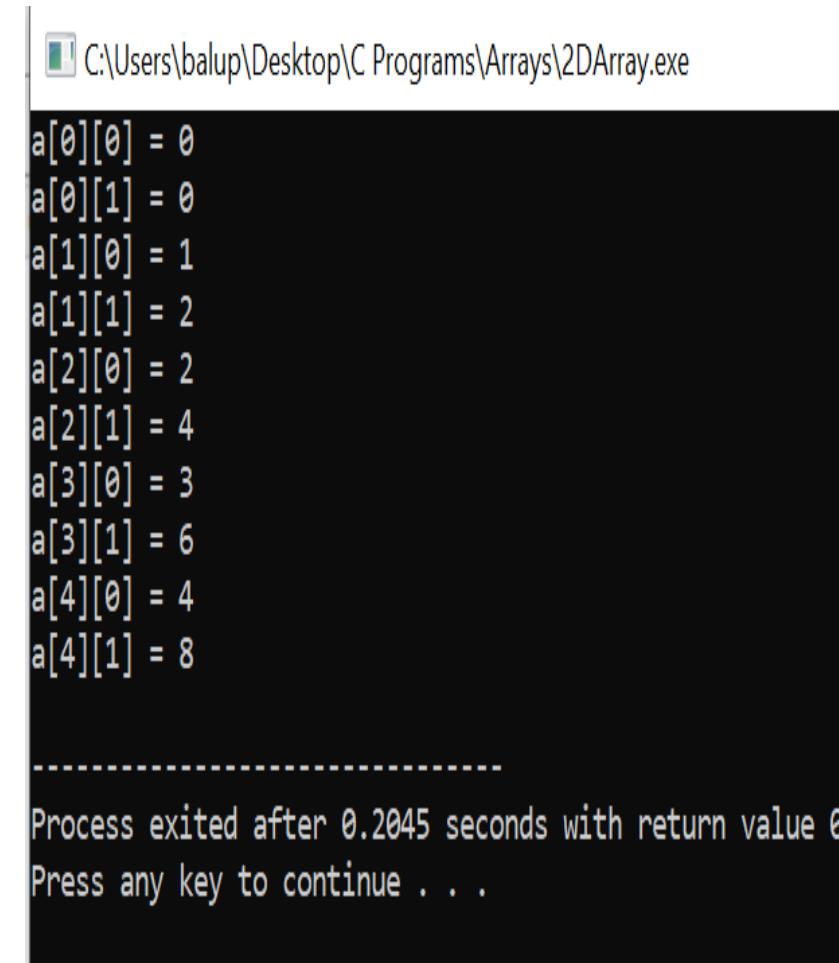
- An element in a two-dimensional array is accessed by using the subscripts, i.e., row index and column index of the array.
- For Example:

```
int val = a[2][3];
```

- The above statement will take the 4th element from the 3rd row of the array.

Accessing Two-Dimensional Array Elements:

```
#include <stdio.h>
int main ()
{
    /* an array with 5 rows and 2 columns*/
    int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8} };
    int i, j;
    for ( i = 0; i < 5; i++ )
    {
        for ( j = 0; j < 2; j++ ) /* output each array element's value */
        {
            printf("a[%d][%d] = %d\n", i,j, a[i][j] );
        }
    }
    return 0;
}
```



```
C:\Users\balup\Desktop\C Programs\Arrays\2DArray.exe
a[0][0] = 0
a[0][1] = 0
a[1][0] = 1
a[1][1] = 2
a[2][0] = 2
a[2][1] = 4
a[3][0] = 3
a[3][1] = 6
a[4][0] = 4
a[4][1] = 8
```

```
Process exited after 0.2045 seconds with return value 0
Press any key to continue . . .
```

Practice Problem:

1. Write a C Program to read and print a row*column Matrix, Row and Column must be input by User.
2. Write a program to read a matrix and prints sum and product of all elements of the two dimensional array.
3. Write a program to print sum of two square matrices.
4. Write a program to find subtraction of two matrices.

A program illustrating 2D arrays...

- A program to multiply the elements of two NxN matrices

How to multiply 2 matrices?

$$\begin{bmatrix} 1 & 2 \\ 2 & 4 \\ 3 & 5 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix} \quad 1 \times 1 + 2 \times 2 = 1 + 4 = 5$$
$$\begin{bmatrix} 1 & 2 \\ 2 & 4 \\ 3 & 5 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix} \quad 1 \times 2 + 2 \times 4 = 2 + 8 = 10$$
$$\begin{bmatrix} 1 & 2 \\ 2 & 4 \\ 3 & 5 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix} \quad 2 \times 1 + 4 \times 2 = 2 + 8 = 10$$

& so on

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 4 \\ 2 & 5 & 3 \end{pmatrix} \quad 2 \times 3$$

$$\mathbf{B} = \begin{pmatrix} 3 & 0 & 4 \\ 1 & 2 & 1 \\ 3 & 4 & 6 \end{pmatrix} \quad 3 \times 3$$

$$\mathbf{C} = \mathbf{A} \times \mathbf{B} = \begin{pmatrix} 3 + 2 + 12 & 0 + 4 + 16 & 4 + 2 + 24 \\ 6 + 5 + 9 & 0 + 10 + 12 & 8 + 5 + 18 \end{pmatrix}$$

$$= \begin{pmatrix} 17 & 20 & 30 \\ 20 & 22 & 31 \end{pmatrix} \quad 2 \times 3$$

A program illustrating 2D arrays

- Consider the data structure that shows the value of sales of three items by four sales girls

	Item 1	Item 2	Item 3
Sales girl 1	310	234	110
Sales girl 2	112	345	321
Sales girl 3	102	321	213
Sales girl 4	250	321	321

- Write a program that computes the following:
 - Total value of sales by each girl
 - Total value of each item sold
 - Grand total of sales of all items by all girls

A program illustrating 2D arrays...

- A program to find the transpose of a matrix

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \quad \text{Original matrix}$$

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}^T \rightarrow \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix}$$