University of Duhok
College of Science
Department of Computer
Second Year

Data Structures
Array and Pointer

Avin J. Kovli
2022 - 2023

OUT-LINES:

- Introduction to data structure.
- Classification of data structure.
- Data types.
- Arrays.
- Pointer.
- Allocation of memory.

INTRODUCTION TO DATA STRUCTURE:

- It is a mechanism of manipulating data. In another words it is a data **organization, management** and **storage format** that enables efficient access and modification.
- Data Structures are widely used in almost every aspect of Computer Science.
  - Operating System, Compiler Design, Artificial intelligence, Graphics and many more.
- Data structures are generally based on the ability of a computer to fetch and store data at any place in its memory, specified by an address.

CLASSIFICATION OF DATA STRUCTURE

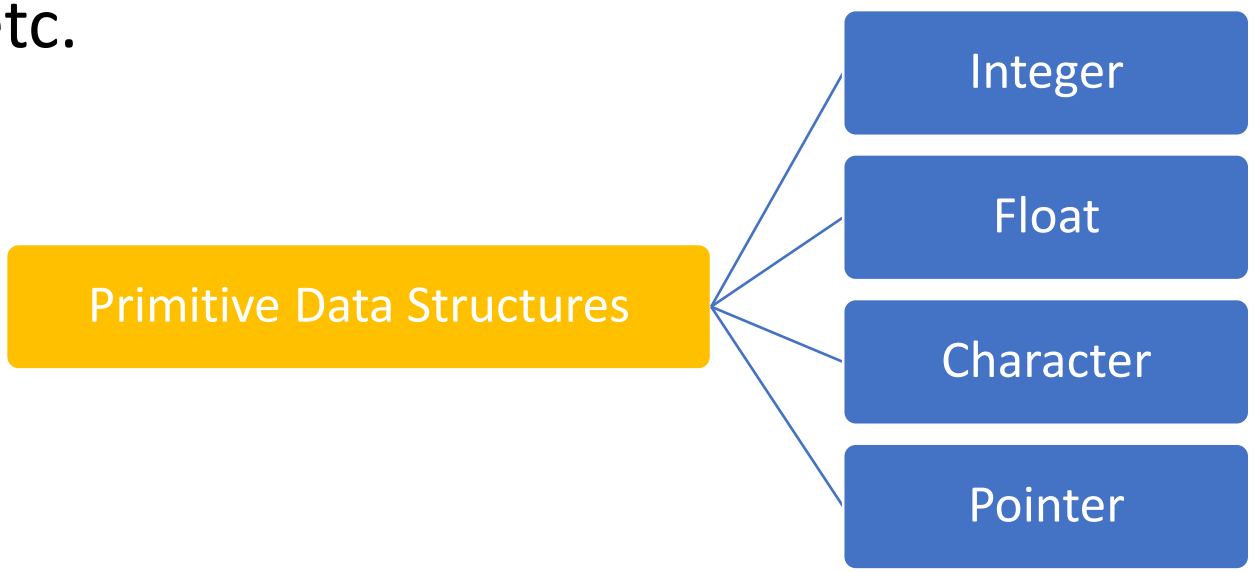
- Data structures are broadly divided into two types :

1. **Primitive data structures :**
2. **Non-primitive data structures :**

CLASSIFICATION OF DATA STRUCTURE

- Data structures are broadly divided into two types :

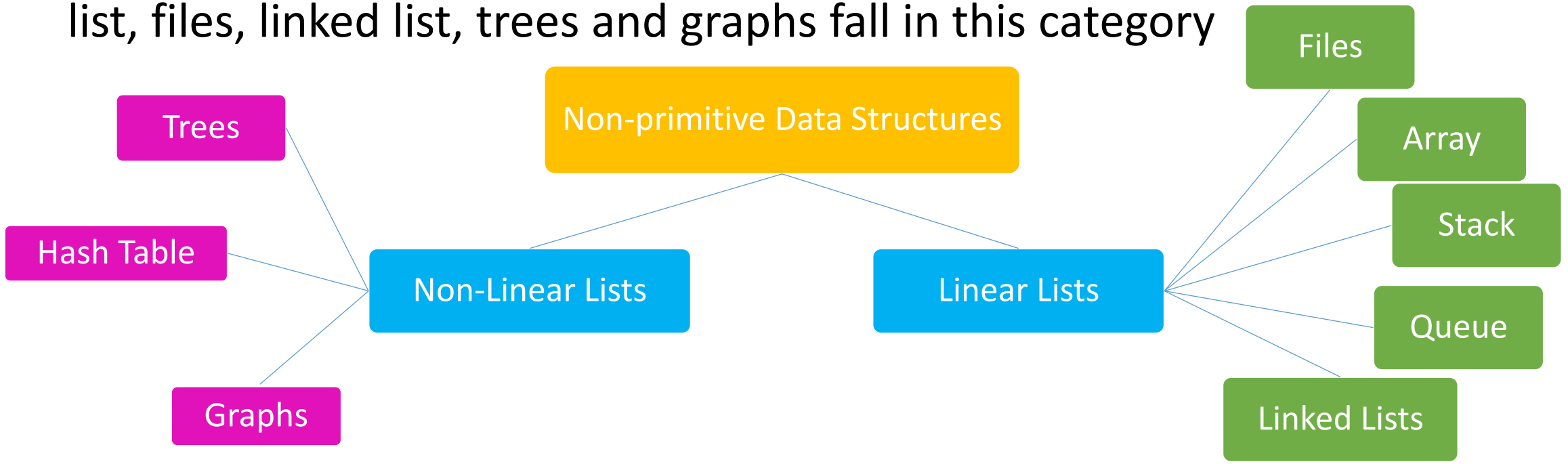
1. **Primitive data structures :** These are the basic data structures and are directly operated upon by the machine instructions, which is in a primitive level. They are integers, floating point numbers, characters, pointers etc.



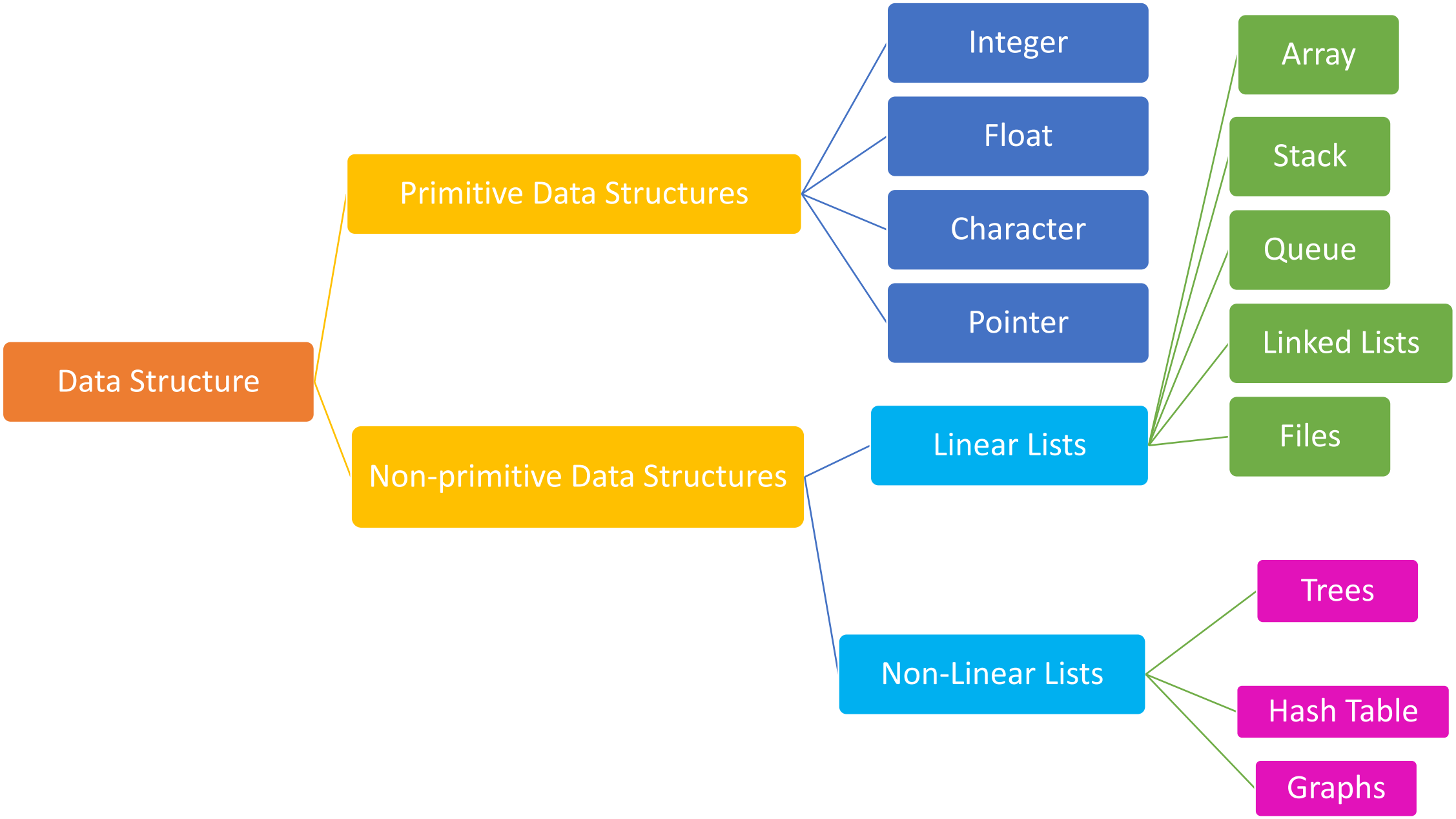
CLASSIFICATION OF DATA STRUCTURE

- Data structures are broadly divided into two types :

2. **Non-primitive data structures :** It is a more sophisticated data structure emphasizing on structuring of a group of **homogeneous** (same type) or **heterogeneous** (different type) data items. Array, list, files, linked list, trees and graphs fall in this category



CLASSIFICATION OF DATA STRUCTURE



DATA TYPES

Data Type	Meaning	Size (in Bytes)
int	Integer	
float	Floating-point	
double	Double Floating-point	
char	Character	
bool	Boolean	
void	Empty	

DATA TYPES

Data Type	Meaning	Size (in Bytes)
int	Integer	2 or 4
float	Floating-point	4
double	Double Floating-point	8
char	Character	1
bool	Boolean	1
void	Empty	0

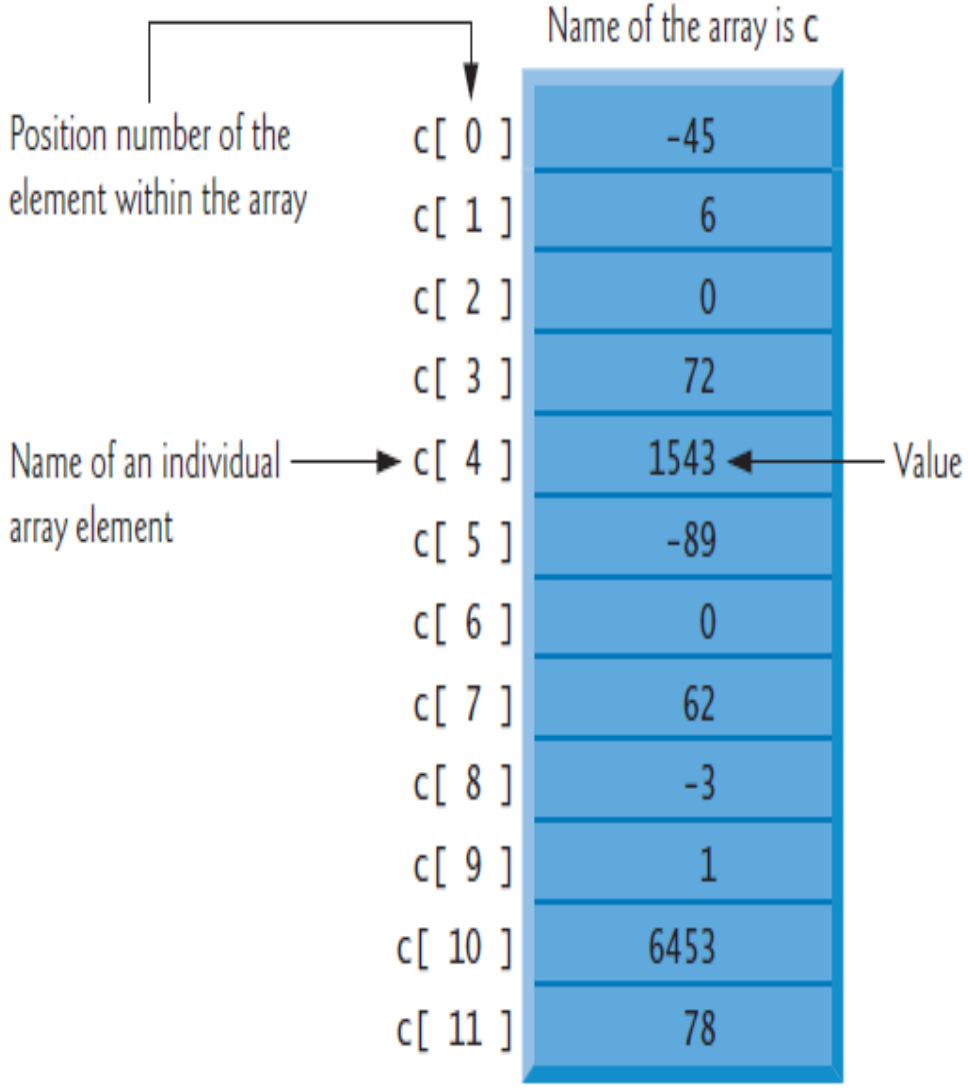
ARRAYS

- An ***array*** is a finite ordered collection of homogeneous data elements that provides direct access to any of its elements.
- ***Finite*** The number of elements in an array is finite or limited.
- ***Ordered collection*** The arrangement of all the elements in an array is very specific, that is, every element has a particular ranking in the array.
- ***Homogeneous*** All the elements of an array should be of the same data type.

ONE DIMENSIONAL ARRAY

- One-dimensional array (or linear array) is a set of ‘n’ finite numbers of homogenous data elements.

- **Declaring one dimensional array**
- **int c[12];** // c is an array of 12 integers
- The ***arraySize*** *must be an* **integer** constant **greater** than **zero**.



EXAMPLE 1:

- Create a C++ program to calculate the sum of array elements of size 5.

```
int main()
{
    int a[5];
    for(int index = 0; index < 5; index ++)
        cin>>a[index];
    int sum= 0;
    for(int index = 0; index < 5; index ++)
    sum = sum + a[index];
    cout<< "sum of array= "<< sum<<endl;
    return 0;
}
```

EXAMPLE 2:

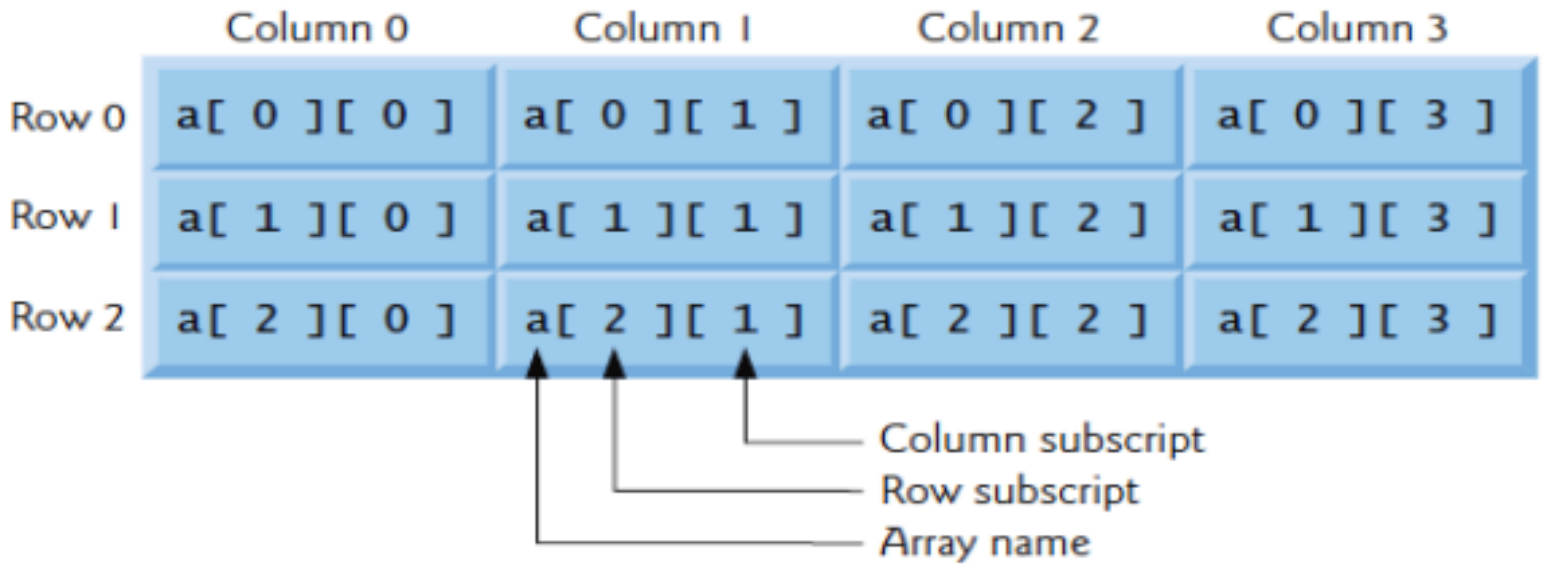
- Write a C++ program to find the maximum and minimum values of an array.

```
int main()
{
    int a[5];
    for(int index = 0; index < 5; index ++)
        cin>>a[index];
    int min=a[0],max=a[0];
    for(int index = 0; index < 5; index ++)
    {
        if(min>a[index])
            min=a[index];
        if(max<a[index])
            max=a[index];
    }
    cout<< "min = "<< min<<endl<<"max= "<<max;
    return 0;
```

ARRAYS WITH TWO DIMENSIONS:

- Arrays with two dimensions (i.e., subscripts) often **represent tables of values** consisting of data arranged in rows and columns.
- By convention, the **first** identifies the element's **row** and the **second** identifies the element's **column**. → **b[2][3]**
- Arrays that require **two subscripts** to identify a particular element are called two-dimensional arrays or **2-D** arrays. Arrays with two or more dimensions are known as **multidimensional** arrays and can have more than two dimensions.
- The array contains **three rows** and **four columns**, so it's said to be a **3-by-4 array**. In general, an array with ***m rows*** *and* ***n*** **columns** is called an ***m-by-n array***.

ARRAYS WITH TWO DIMENSIONS:



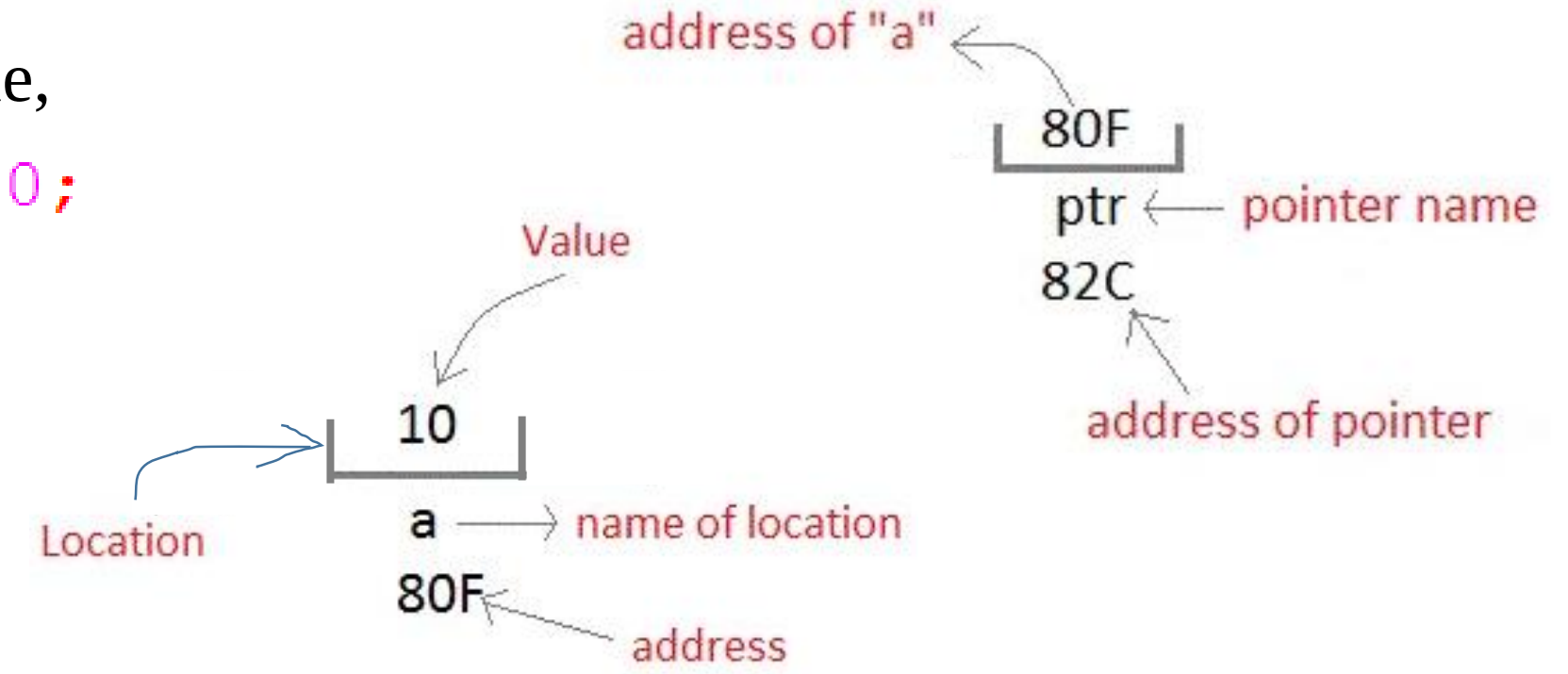
ARRAYS WITH TWO DIMENSIONS:

- Initialization of multidimensional Array can be initialized in its declaration much like a one-dimensional array.
- int b[2][3] = { { 1, 2,5 }, { 3, 4,6 } };

POINTER

- A **Pointer** is a special variable because it's data (value) is a **memory address** of another normal variable (mostly) of **same data type**.
- Pointers are **used to access memory** of a variable and manipulate the value stored in it.
- For Example,

```
int a=10;
```



DECLARE A POINTER:

- To declare a pointer variable, you must specify the type of value that the pointer will point to.
- for example,

```
int* ptr;// ptr will hold the address of an int
char* q; // q will hold the address of a char
```

- To point to a variable;

```
ptr=&a;
```

POINTER OPERATORS IN C++:

1. * Operator:

This operator returns the **value** located at the given address.

2. & Operator:

The & operator returns the **memory address** of its operand.

- The * operators is the complement of &.

EXAMPLE 1:

```
int x= 25;
int y=120;
bool c=true;
bool* ptr;
ptr=&c;
cout<<&x<<endl;
cout<<&y<<endl;
cout<<&c<<endl;
cout<<&ptr;
```

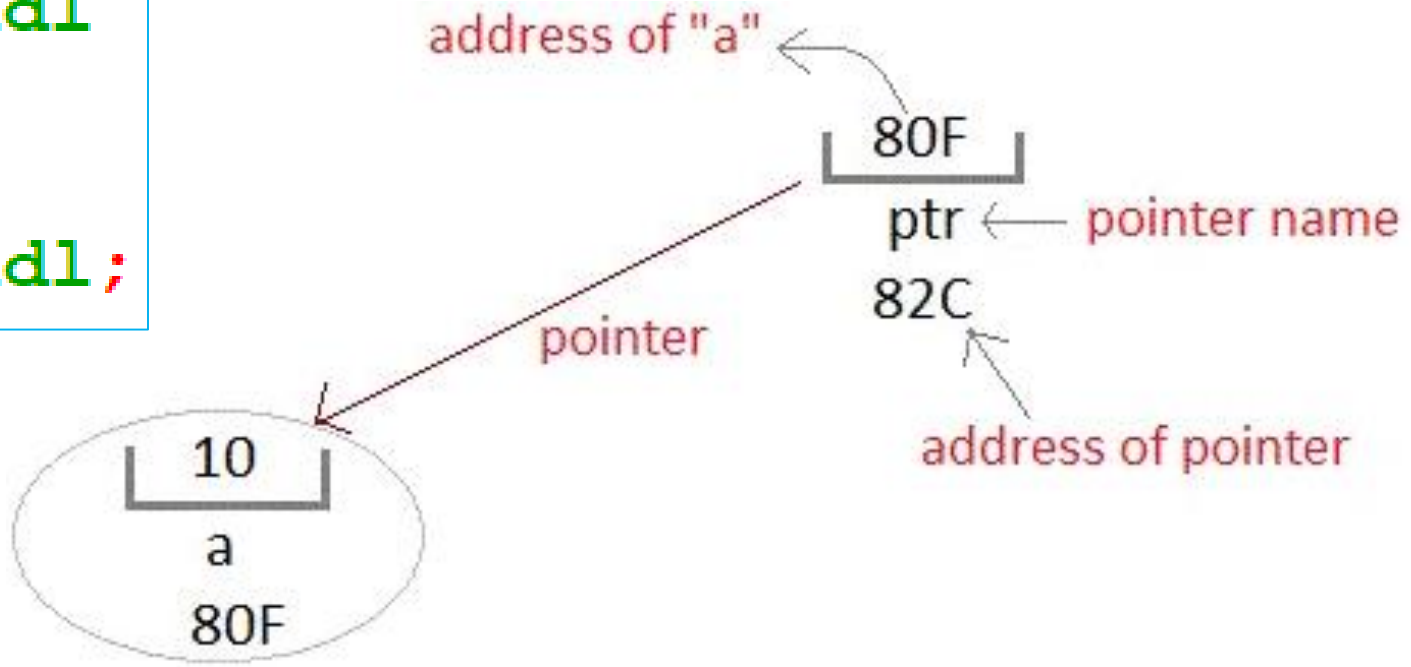
VarName	Address	Value
c------	0X010B	1(True)
		: :
ptr------	0X02FF	0X010B
x--------	0X0300	25
: :		: :
y -----	0X8080	120

EXAMPLE 2:

```
int a=10;
int* ptr;
ptr=&a;
cout<<"a= "<<a<<endl
<<"*ptr= "<<*ptr<<endl
<<"&a= "<<&a<<endl
<<"ptr= "<<ptr<<endl
<<"&ptr= "<<&ptr<<endl;
```

Output:

```
a=     10
*ptr=  10
&a=    80F
ptr=   80F
&ptr= 82C
```



HOW TO MAKE A POINTER POINT TO AN ARRAY C++?

- A pointer not only able to store the address of a single variable, it can also store the address of cells of an array.
- For example;

```
int *ptr;
int arr[5]={22,33,44,55,66};
ptr = arr;
```

arr[0]	arr[1]	arr[2]	arr[3]	arr[4]
22	33	44	55	66

←is equivalent to this→

```
ptr = &arr[0];
```

- The code **ptr = arr;** stores the address of the **first element** of the array in variable **ptr**.
- Suppose we need to point to the **fourth** element of the array using the same pointer **ptr**.

```
cout<<*(ptr+3)<<endl;
```

HOW TO MAKE A POINTER POINT TO AN ARRAY C++?

```
char *pc;
char c[]="computer";
pc=c;
cout<<pc<<endl;
 char *pc;
 pc="computer";
 cout<<pc<<endl;

cout<<*(pc+2)<<endl;
```

Any questions?

THANK YOU