DATA STRUCTURE AND ALGORITHMS PART - 1

TOPICS FOR TODAY

1. Arrays
2. Linked Lists
3. Doubly Linked Lists
4. Binary Tree
5. String Interning

LET'S BEGIN

WHY DATA STRUCTURE AND ALGORITHMS ?

TYPES OF DATA STRUCTURE

a. Linear Data Structure
b. Non-Linear Data Structure

ARRAYS

WHAT IS AN ARRAY?

finite ordered set of homogeneous elements.

Finite: specific no of elements, be it small or large

ordered: all elements are arranged so that there is 0th, 1st, 2nd elements.

homogeneous: all elements must be of same type

C
DECLARATION

1. By specifying size

Syntax:

DataType varName[size]

Example:

int a[100]; // array of 100 integers, contains garbage value

C
DECLARATION

2. By Initializing Elements

Syntax:

DataType varName[] = {data1, data2, ...,n-1 }

Example:

int a[] = {1,2,3,4,5}; // array of 100 integers

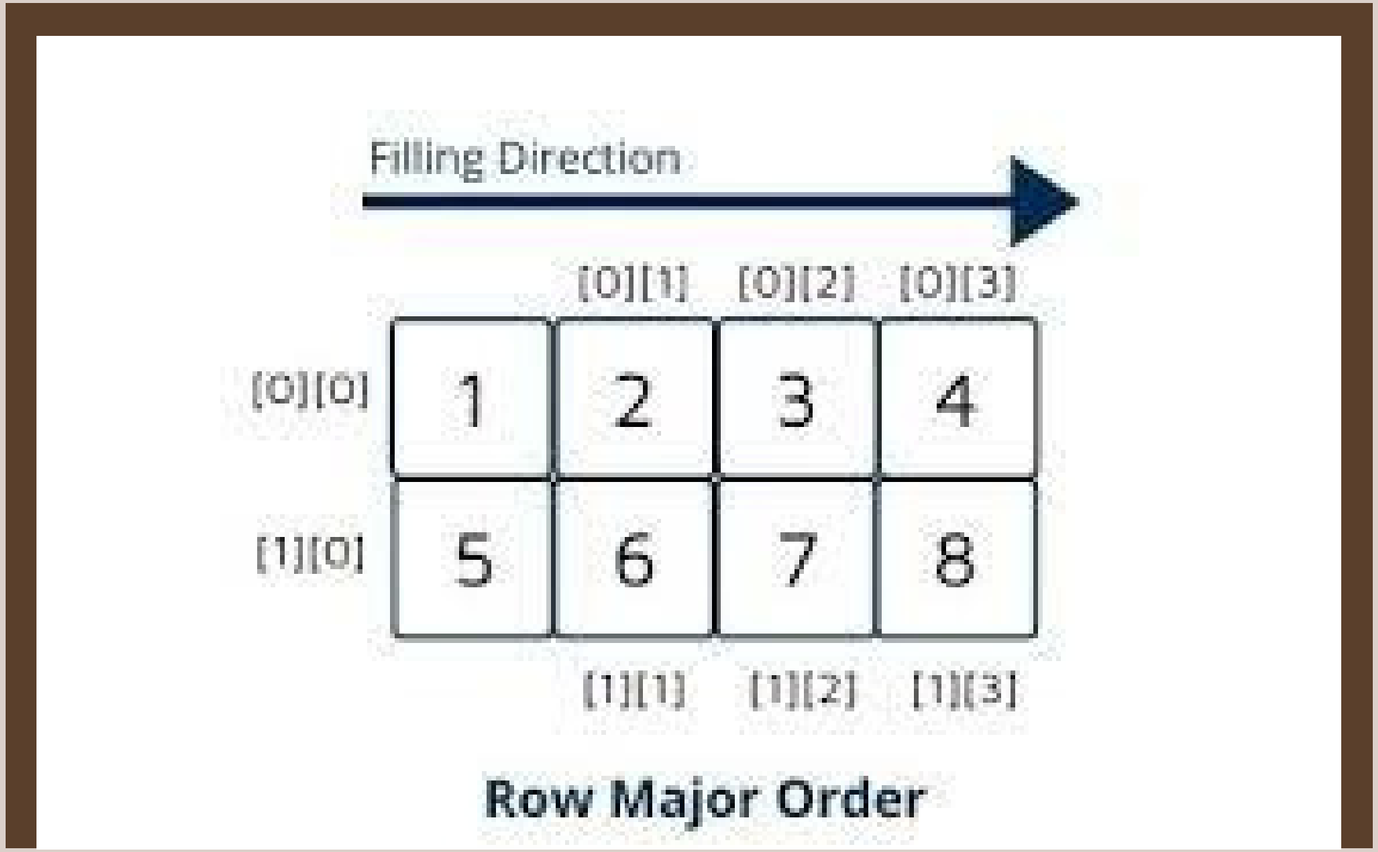
HOW IS AN ARRAY STORED IN MEMORY?

Two Dimensional Array

- Component of one array is another array.
- Logical data structure useful in programming and problem solving (ie. it is easier to describe an object that is physically 2 dimensional, eg. map, checkerboard etc).
- But the memory of the computer is linear. So there are many ways to represent a 2d array in memory.

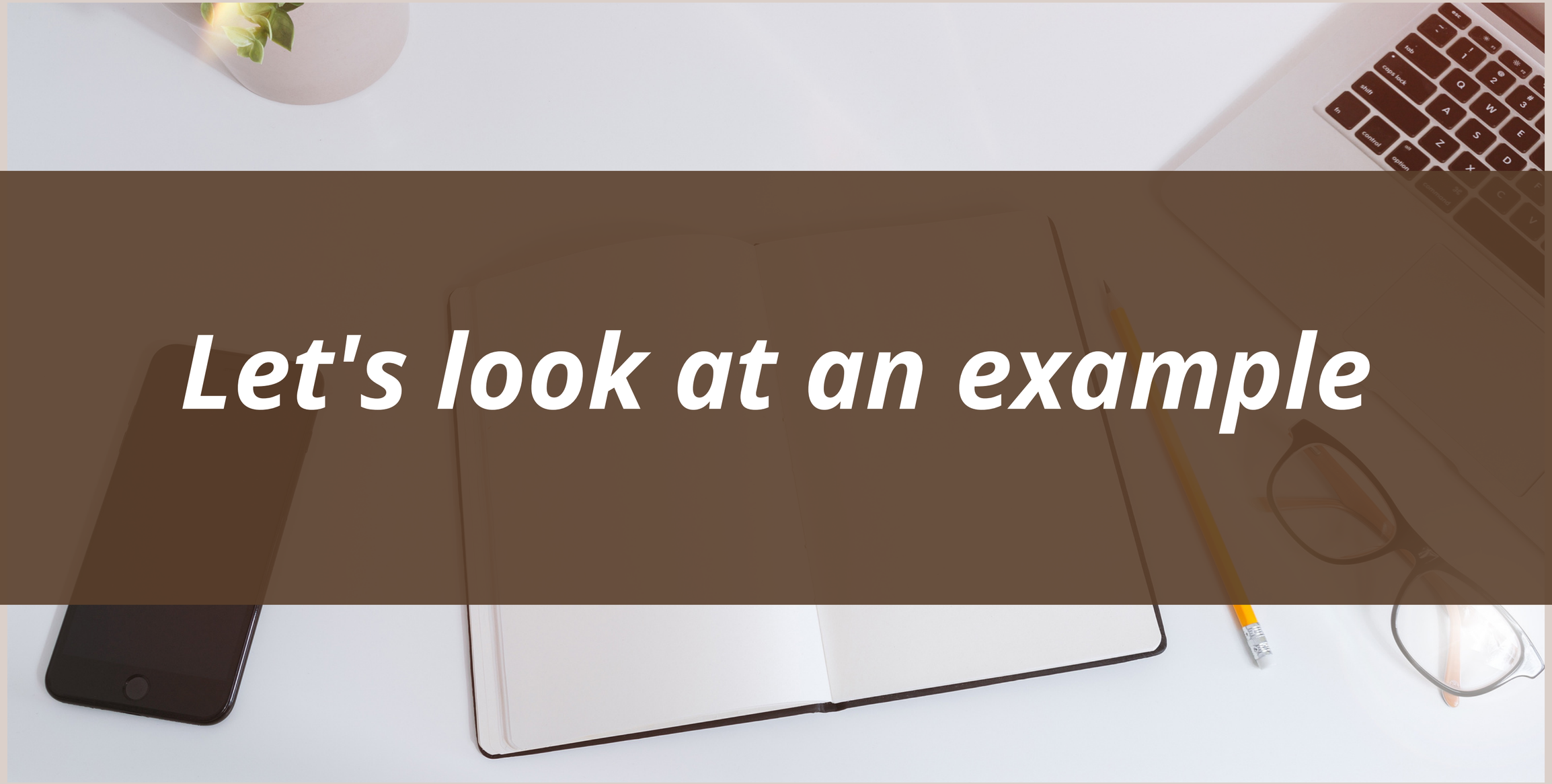
Row-Major Approch

address of a[row][column]: base addr + (width*row) + column



DYNAMIC ARRAY

Dynamic arrays are resizable and provide random access for their elements. They can be initialized with variable size, and their size can be modified later in the program. Dynamic arrays are allocated on the heap



Let's look at an example

BASIC ARRAY OPERATIONS

- Traversing
- Searching
- Inserting
- Deleting
- Sorting
- Merging

LINKED LISTS

ARROW STRUCTURE

foo->bar is equivalent to (*foo).bar, i.e. it gets the member called bar from the struct that foo points to.

GET VERRRY COMFORTABLE WITH IT.

SOMETHING ABOUT STRUCTURES.....

```
struct name1 {
    int data;
    };
```

```
struct name2 {
    int data;
struct name1 s;
    };
```

```
struct name4 {
    int data;
struct name4* s;
} ;
```

LINKED LIST

```
struct node {
int data;
struct node* s;
} ;
```

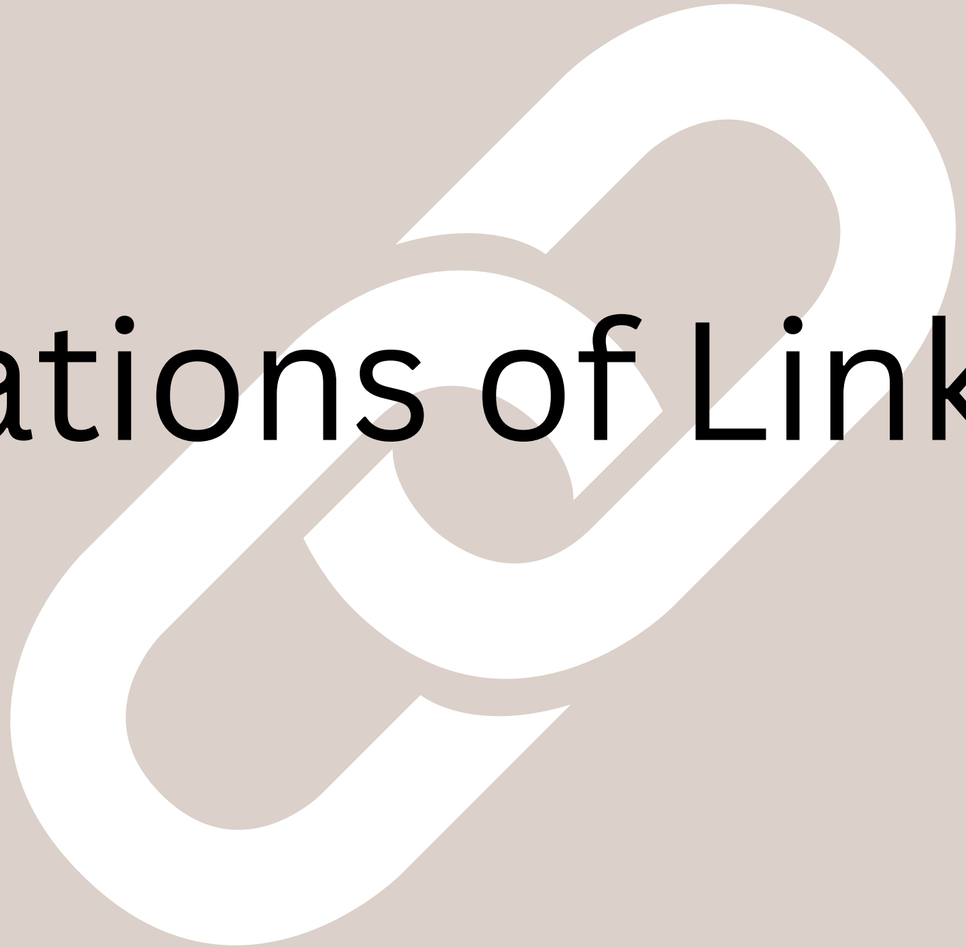
The member s allows us to do something clever. Chain the structures!

Implementation in C
Given as an assignment..

DOUBLY LINKED LISTS

```
struct dbl_ll {
int val;
struct dbl_ll* next;
struct dbl_ll* prev;
};
```

Applications of Linked List



DISADVANTAGES

1.Cache Misses.

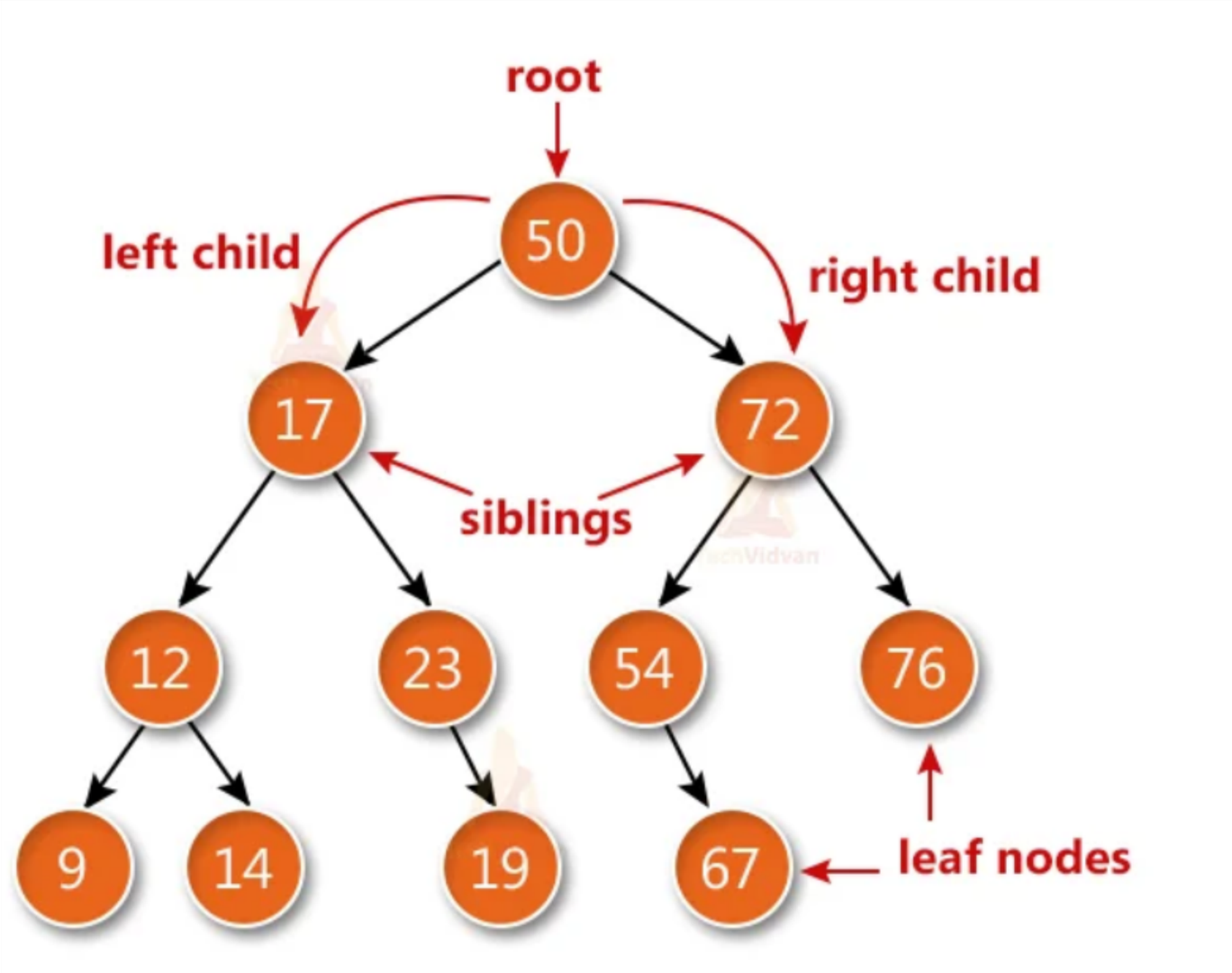
2. You can accidentally loose pointer. Will be problematic if it was heap memory.

No way to go back a node.

BINARY TREE

Terminologies

1. Root
2. Siblings
3. Internal Nodes
4. External Nodes(Leaf Nodes)
5. Ancestors
6. Descendants
7. Edge
8. Height
9. Depth



STRUCTURE

Linked list used to point to one another element. How 'bout a little more?

```
struct Binary_Tree {
int val;
struct Binary_Tree* left;
struct Binary_Tree* right;
};
```

Application of Binary tree

1. Decision Tree
2. Searching
3. Data Compression
4. Expression Trees
   (A B C*+ D/)

5. etc.

Try implementing a binary search tree of the data given in an array.

STRING INTERNING

“String interning” (sometimes called “string pooling”) is an optimization that consists of storing only one copy of a string, no matter how many times the program references it.

```c
#include <stdio.h>

int main()
{
    char* a = "Anju";
    char* b = "Anju";
    char c[] = "Anju";
    char d[] = "Anju";
#define print(x) printf(#x " = %p\n", x)
    print(a);
    print(b);
    print(c);
    print(d);
#undef print
}
```

OUTPUT?

```
Acer in trash ❯ .\mem_model.exe
a = 00007FF6CDCAD000
b = 00007FF6CDCAD008
c = 000000106B1DF860
d = 000000106B1DF868
Acer in trash ❯ .\a.exe
a = 00007FF6AB6C32F0
b = 00007FF6AB6C32F0
c = 000000F8046FF6E3
d = 000000F8046FF6DE
```

THANK YOU