بسم الله الرحمن الرحيم

data structures

↳ data organization

obj:-

Processing large amount of data in short time

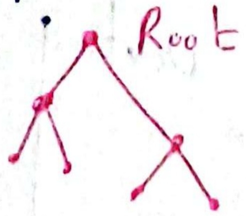
Reduce space & time Complexity

memory

Types of data structures

- Arrays ↓ Contiguous
- stack FiLo First in Last out
- heap ↓ Dynamic memory Allocation
- Queue fiFo First in First out

- Tree



- Tables
- Linked lists → stack / QuQue implementation
- Graphs

static & dynamic data structures

static ↓ Fixed size. Ex: Arrays

dynamic ↓ Can be modified during run-time. Ex: Linked lists

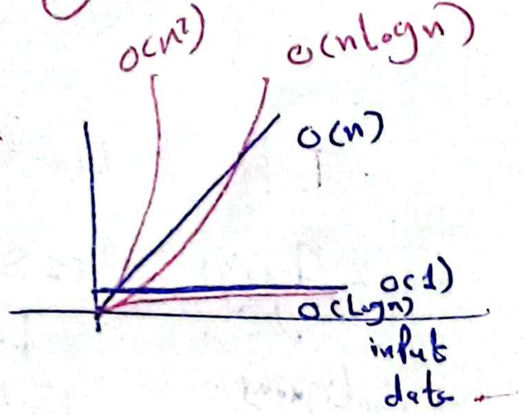
Time & space Complexity

→ How much memory allocated during Run-time

→ How much Time to modify data structure

iterations / steps to finish Program

— Big-O Notation Vs data size



Best: [O(1) : independent

[O(Logn) : Logarithmic

O(n) : Linear

$\boxed{O(n\log n) \ - O(n^2)}$ Worst XXX

	Access	Search	insertion	Deletion
Array	o(1)	o(N)	O(N)	o(1)
stack			o(1)	o(1)
Queue	O(N)		o(1)	o(1)
Linked List			o(1)	o(1)

Stack → [size – top] ↙ characterstig

↳ Last in First out LiFo or FiLo

adding to the top → Push
Removing From the top → PoP

stack[n-1]
stack[0]
-1

① Push : top is < -1 For empty stack ; (n-1) For full stack

n → No. of elements

↳ check if stack is full

- if full → stack overflow
- not full → increment top by 1, store data into stack Array

② PoP : <
- if empty → empty stack error
- Not empty : read data, decrement ~~data~~ top by (1)

Applications : Keyboard, undo-Redo
History
Call Logs & emails & gallery

* implementing stack

struct {array, integer}

↓ stack[size] ↓ top

* Pop & Push functions, is empty, is full.

Prototypes.

```
typedef struct stack {int elements[Size]; int top};
void CreateEmptystack (stack*) -> set stack top to -1
```

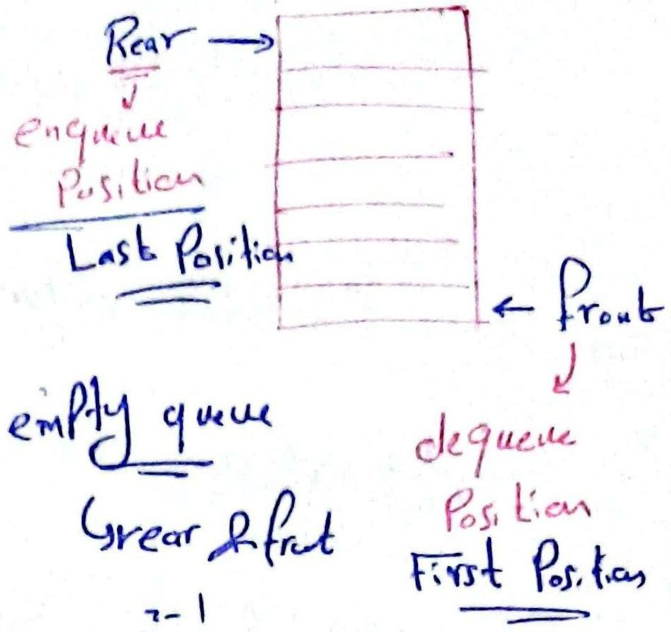
$\boxed{\text{int}}$ Push (stack*, int data);
↳ Error

$\boxed{\text{int}}$ Pop (stack*, data*)
↳ Error

$\boxed{\text{int}}$ Printstack / isFull / is empty / get Top (Stack*)
| Error

Error: $\boxed{\text{int}}$ or Boolean

Queue → All should go to add new Element $\underline{o}$



→ Enqueue

if full → Full error

not full & empty
→ increment rear & front

not full & not empty
→ increment rear

empty queue
rear & front
= -1

→ de queue

if empty → empty error

not empty & last element : -read - set rear & front by -1

not empty & not last element: read - increment front

Applications :- scheduling & uploading
downloading

implementing queue:-

struct { array, rear, front }
integers

Functions: enqueue - dequeue

Prototypes

typedef struct queue {int elements [size] ;
int front; int size; }

void createEmptyqueue (queue*)

enque (~~stack~~ q* , data)
deque (~~stack~~ q* , data*)

Error: int or Boolean

Helper Functions:

Print queue
get Queue front
get Queue Rear
is Full
is Empty
(queue*)

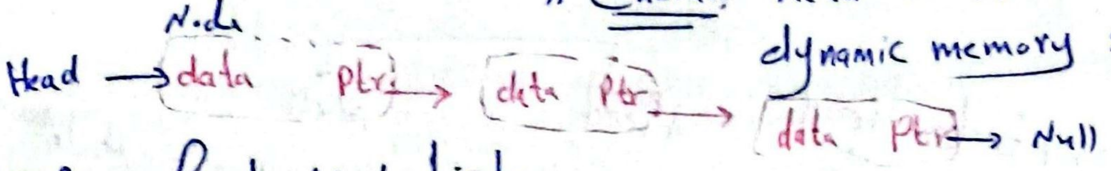
Error State

Linked List

↳ Connect data Nodes together without need of Contiguous Sorting [Pointers]

Ex: All + taB

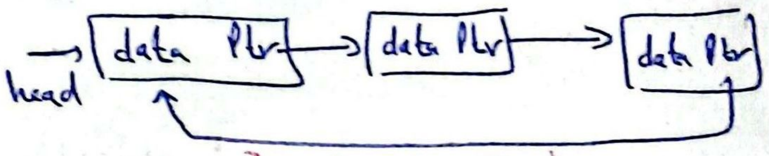
* Charact head & Tail
dynamic memory size



Types of Linked lists:

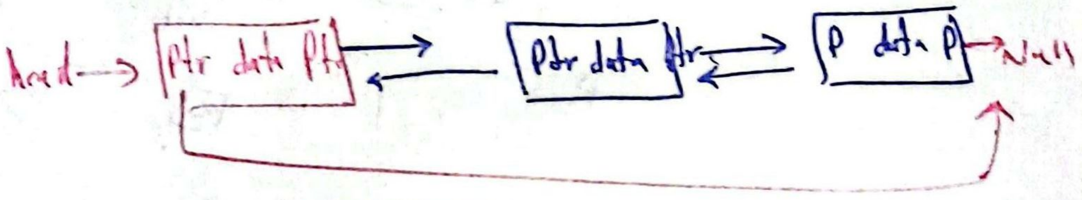
* Single Linked list: one Pointer for a node
Last Node Points to Null

* Circular Linked list:

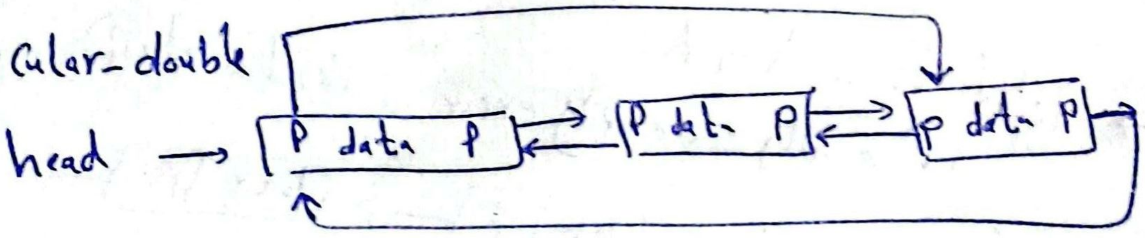


Last Points to First

* double_Linked List → two Pointer in a node



* Circular-double



* Insert & delete & Print

struct Node

ptr next
data

struct representation

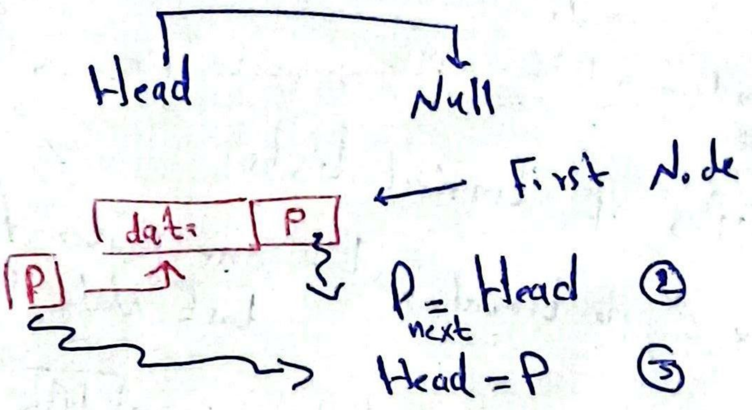
ptr head
int size

empty → head = Null

linked-list → in heap - dynamic memory allocated

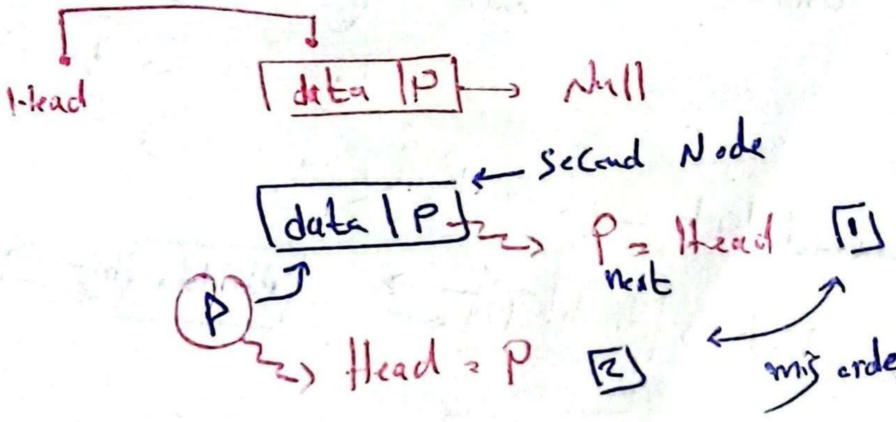
→ insert to head

① allocation



P_{next} = Head ②

Head = P ③



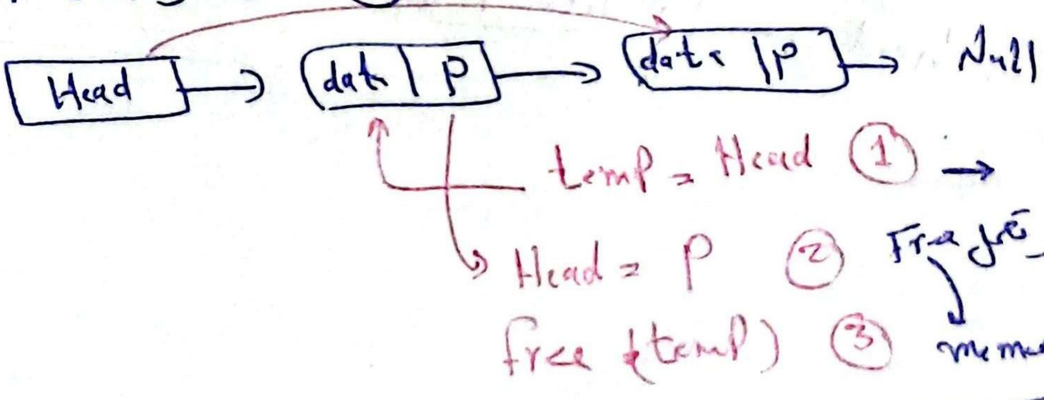
P_{next} = Head $\boxed{1}$

Head = P $\boxed{2}$

Warning
mis order Wastes data

→ delete from head

if empty → empty error (Head = Null)



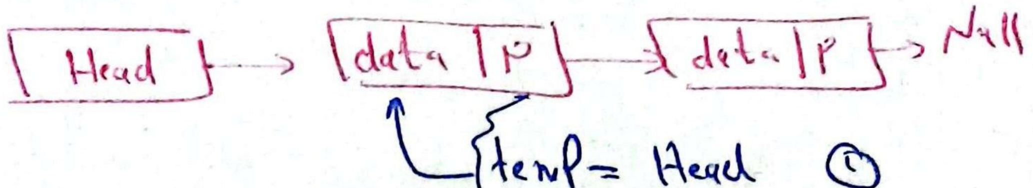
temp = Head ① → [illegible]

Head = P ② عشان تقدر تعمل Free

free (temp) ③ memory Leaks

→ Print List

head → Null ⇐ is empty / Last Node ⇒ P = Null



if p ≠ Null : { temp = Head ①
Print ②
temp = p ③ }

P = Null → Print & stop

Applications : Stack & Queue implementation
Photo Viewer
Switching apps

---

implementing a Linked List.

ptr head   struct node {data, ptr next}

insert & delete functions
is Empty , is Full , is Tail , Print List ← Helper Functions