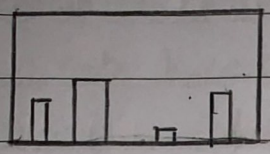
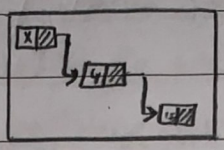
الموضوع: _____ التاريخ: / /

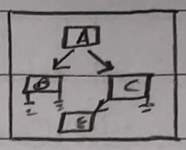
Data Structure : الطريقة التي من خلالها نقدر نجمع مجموعة من البيانات وننظمها لإجراء العمليات بطريقة فعالة



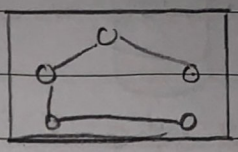
Sorting



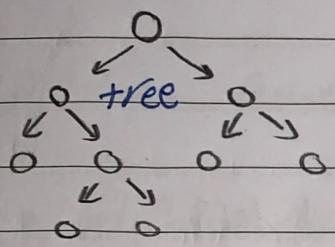
Linked List

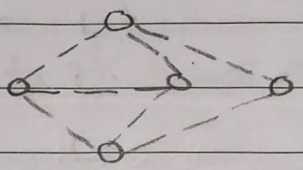


List

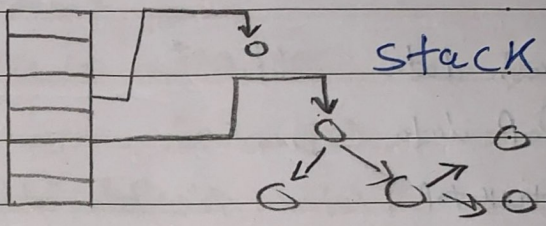


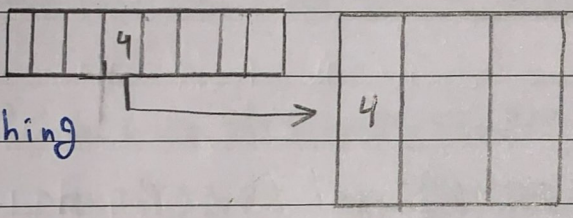
Spaning tree





Graph





Hashing

1. stack : is a linear list data structure

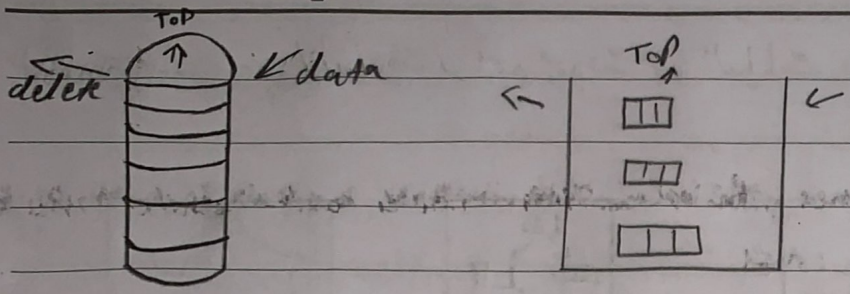
* which all addition and deletions are restricted to one end, called the top.

* if you insert a data into a stack and then remove it, the order of the data is reversed.

* Data input [0, 5, 10, 15, 20] is removed as [20, 15, 10, 5, 0]

* this reversing attribute is why stacks are known as "LIFO Data structure" the last in first out

التاريخ: / / الموضوع: _____



stack of Coin Computer stack

Basic stack operations:

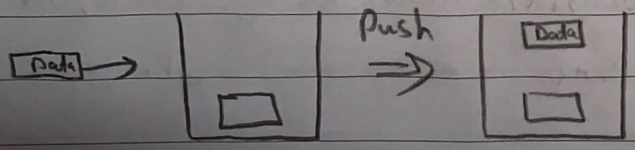
1- Push data "adding"
2- PoP data "removing"
3- stack top "return the data at the top of the stack without removing it"

mendatory stack operations:

1- stack intialize: "intialize the stack data type and stack Pointer
2- stack size: "Number of elements in the stack / Num of elements * size of one element"
3- stack Display: "Display the stack content start with the top"
* You can add any helper function as you need

Push data: adding new item to the stack

* after Pushing, the new item Become the top.
* the only Potential ProPlem with simple operation it must ensure that there is room for the new item "stack over flow" (You can use helper function to ensure that)



التاريخ: / / الموضوع: _____

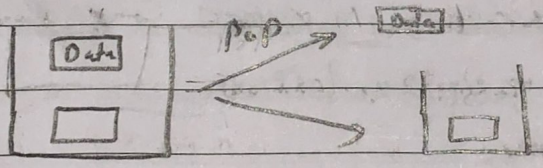
Push stack Pseu code:

1- Pre: my stack Passed by refrence, the Value that Contained data Pushed into the stack
2- Post: Data have been Pushed in the stack.
3- Procedure: Validate if the stack not full, return error if it is full
* increment stack Pointer
* update the Location by the data Provided from user.

* ——— *

Pop data: removing data from stack, return it to the user.

* after Poping, the next older item becomes the top.
* when the last item in the stack deleted the stack must be it's empty safe "underfl
* if Pop called when the stack is empty, it is in underflow state (we can use helper funct

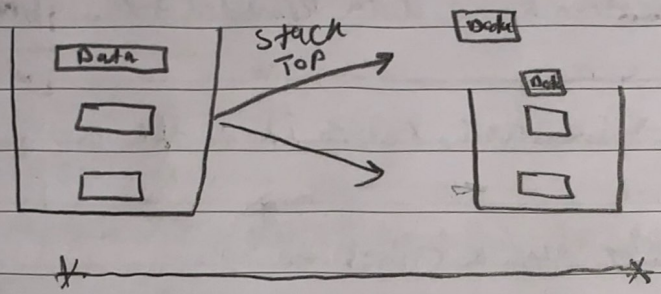


Pop Pseu code:

1- Pre: stack Passed by refrence, value is refrenced Variable to recive data
2- Post: Data have been returned by calling algorithms, Return true if succesful, false if underflow
3- Procedure: validate the stack not empty
* if stack empty "Can't Pop"
* set Value into Pop node
* Decrment Stack Pointer

stacktop : copies data in the top, return it to user

* you might think of this operation as reading the Stack top.
*
* stack top can also result in underflow if the stack is empty, so we need to verify if stack is empty.



stack top Pseudocode:

1- Pre : stack Passed by reference, value is refrenced variable to recive data
2- Post : Data have been returned by calling algorithm
* return true if data returned, false if underflow
3- Procedure : * validate if stack is empty.
* return error if stack is empty.
* set value to data in top node.

helper functions : 1- stack initialize : stackPtr = -1 ; / count = 0 ;

2- stack empty : check if stack empty "boolean" → true if "it's empty"
→ false if "it's Not Empty"

3- stack full : , , , , , , full "boolean" → false if "it's Not full"
→ true if "it's full"

4- Stack Count : return stack Pointer + 1

* stackPtr = -1 stack is empty
* stackPtr = Max-1 stack is full

التاريخ: / / الموضوع: _____

Static Stack : stack maximum size can be calculated before the program is written, an array implementation of a stack is more efficient than Dynamic implementation

Dynamic Stack : stack maximum size calculated while the run time

structure of Dynamic Stack
- **stack array → Dynamic memory لتمثيل object Stack لـ جميع الـ stack وبعدها تمثل Double Pointer لـ Dynamic memory بالنهج الى انشاءاوزه
- Count
- Stack_max
- top

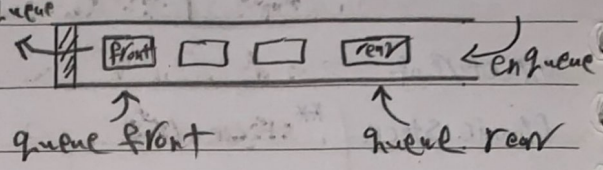
structure of static stack
- top
- stack array [Stack_max]

التاريخ: / / الموضوع: _____

Queue Data Structure: a linear Data structure,

* which Data can only be inserted at one end, called rear and the other called front

* these restrictions ensure that the data are Procesed through the queue in the order in, which they are recived



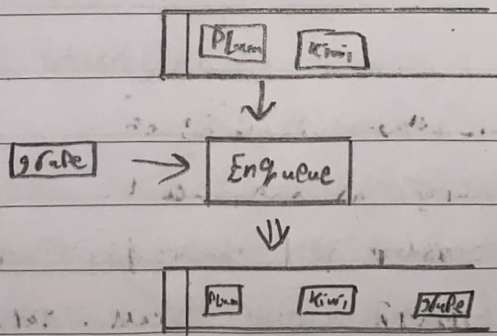
* in other words, a queue is a first in first out "FIFO" structure.

* queue is the same as a line "like waiting Jobs to be Processed by a Computer is a queue

* queue operations:

1- Data can be inserted at rear "Enqueue"
2- Data can be removed from the front "Dequeue"
3- Retrived from the front "queue front"
4- Retrived from the rear "queue rear"

1- Enqueue, the queue insert operation



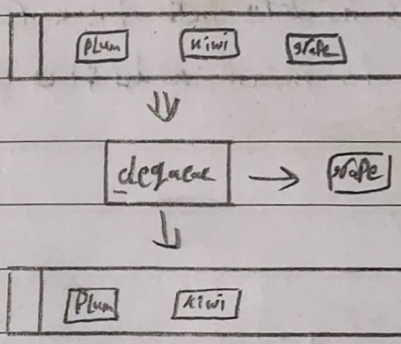
* after inserting data into the queue the new element become the rear
* the only Potential Problem with enqueue is runing out of room for the data if there isn't enough room for another element in the queue, the queue is in an overflow state

التاريخ: / / الموضوع: _____

EnQueue Pseudocode:

1- Pre: queue_object by refrence, itemPtr Contain data tobe inserted into the queue "rear side"
2- Post: item_Ptr has been inserted
3- Procedure: validate if the queue isn't full
* return error if queue is full
* increment the queue front
* increment the number of elements
* update the new Location by the data provided by the user.

DeQueue: the queue delete operation



* the data at the front of the queue returned to the user and removed from the queue

* if there is no data in the queue when a dequeue is attempted, the queue is in an underflow state.

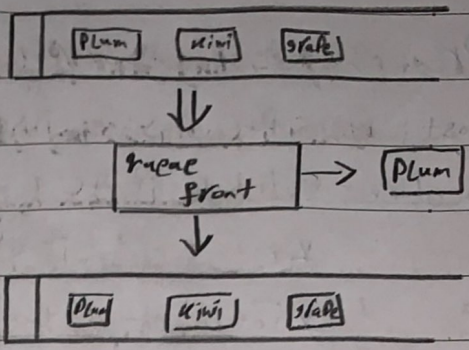
DeQueue PseuCode:

1- Pre: queue_obj by refrenced
2- Post: item_Ptr has been deleted
3- Procedure: validate if the queue isn't full
* return error if the queue is empty, increment queue front Ptr.
* update the new Location by the data provided by the user
* decrement the number of element

الموضوع: _____ التاريخ: / /

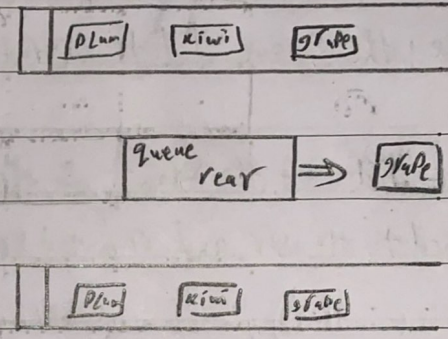
Queue front: return the data at the front of the queue without removing it.

* if there is no data in the queue front is attempted, the queue is in an "underflow state".



Queue rear: a Parallel operation to queue front return queue rear without removing it.

if there is no data in the queue rear is attempted, the queue is in an "underflow state"



* في حالة لو جاوز اخر عنصر في queue rear = Max-1

في حلين:

1- shift all elements from the end to the begining of the queue
↳ "forbiden solution because it use more processing solutions"

2- using circular queue:

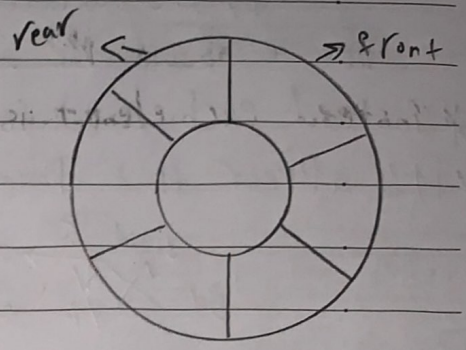
* Queue is full — Couter = Max — front = 0
Queue is empty — Couter = 0 — front > rear

التاريخ: / /

الموضوع: _____

Circular queue : is a queue the last element is logically followed by the first elem

* this done by testing for the last element rather than adding one, setting the index to zero



	0	1	2	3
f=-1 r=-1	–	–	–	–
f 0 r=0	1	–	–	–
f=0 r=1	1	2	–	–
f=0 r=2	1	2	3	–
f=0 r=3	1	2	3	4
f=1 r=3	–	2	3	4
f=2 r=3	–	–	3	4
f=2 r=0	5	-	3	4
f=2 r=1	5	6	3	4
f=3 r=1	5	6	-	4
f=0 r=1	5	6	-	-
f=1 r=1	-	6	-	-
f=-1 r=-1	-	-	-	-

Max=4
Counter=0

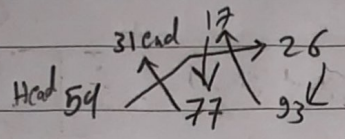
Enqueue
if (rear = Max)
↳ rear=0

Dequeue if (front=Max)
front=0

if (front = rear) {
front=-1
rear=-1
Counter=1}

Linked List: linear collection of Data elements whose order is not determined by the replacement in memory.

* instead each element is in a node which Points to another node.
* it's appear that these values have been placed randomly.



types of Linked Lis:

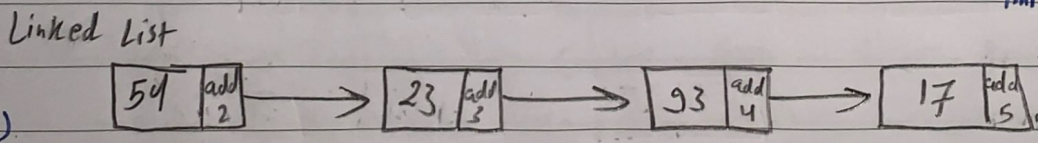
1. unordered Linked List: data stored in not order inside the list
2. ordered Linked List: data stored in order inside the list.

Basic operations:

1. insertion: adding elements to the list, at the (beginning, between, end)
2. Deletion: removing elements from the list, from (beginning, between, end).
3. Retval: retrive data from the list or determine data Position.
4. Traversal: Processes each element in a list in sequence.

---

* each node object must hold at least two Pieces of information:
  1. first, the node must Contain the list item itself => Data field.
  2. Second, each node must hold a reference to the next node Link field.



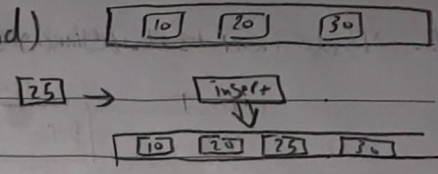
implementation:

```
typedef struct node {
    int Data;
    struct node * Link;
} Node;
```

التاريخ: / /

الموضوع:

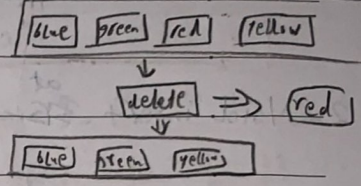
1. insertion: adding element to the list at (beg, bet, end)



*ordered list are maintained in sequence according to the data or key that identifies the data

~~*Examples: the key~~

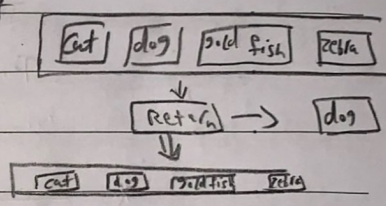
2. Deletations: Removing element from the list from (beg, bet, end)



*once located, the data are removed from the list.

*fisure depicts a deletion from list

3- Retrival: Return data from the list, or determine the Position



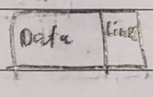
4. Traversal: Processes each element in a list in sequence

* it's require looping algorithm rather than a search.

* the Loop term when all elements have been processed.

Basic Linked List types:

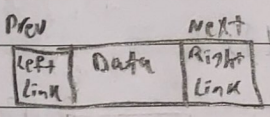
1- Single Linked List:




```
typedef struct node {
    uint32 Data;
    struct node *Link;
}
```

2- Double Linked List:

3- Circular Linked List:

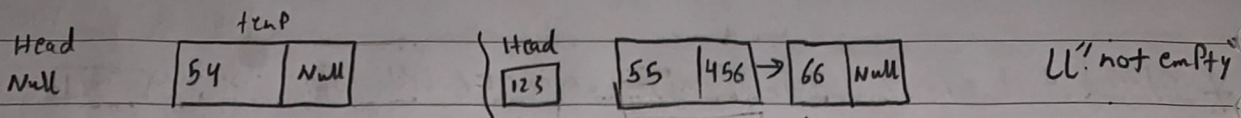



```
typedef struct node {
    uint32 Data;
    struct node *Left;
    struct node *Right;
};
```

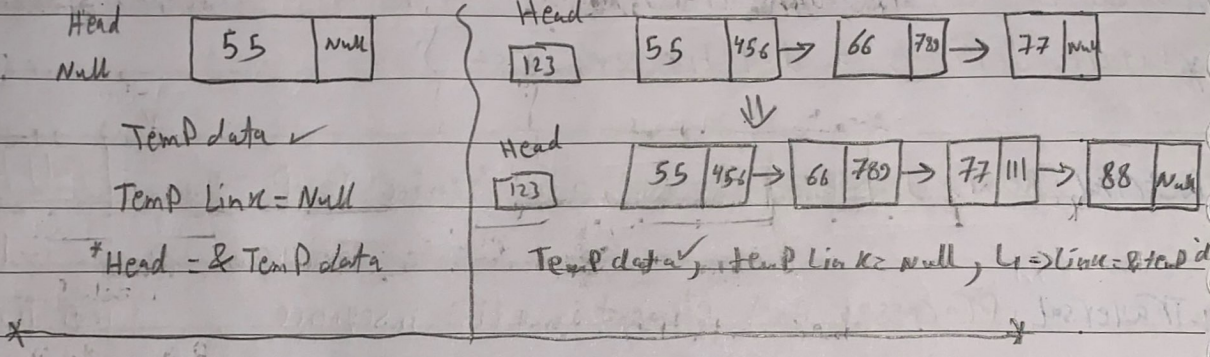
*to pass a linked list to a function this function will change the linked list Head, you must pass it by refrence

التاريخ: / / الموضوع: _____

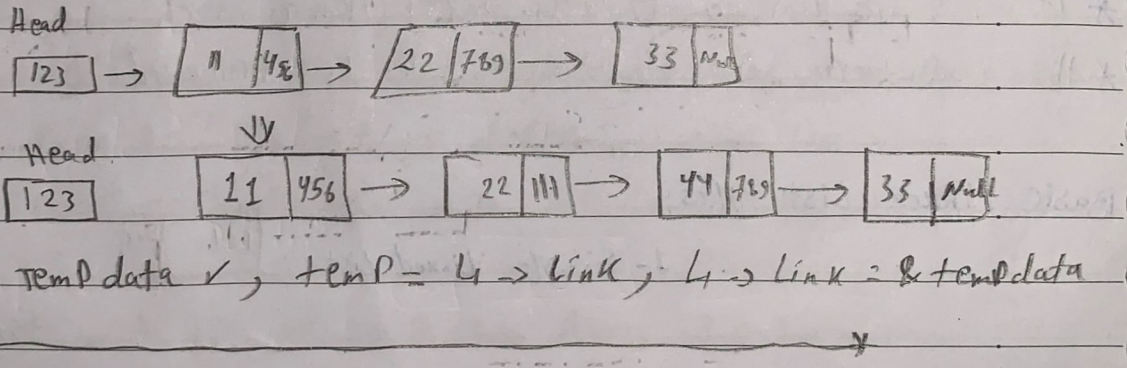
1- Void insert_at_beginning (Node ** Head); "RL_LL" ⇒ Right Link, Left Link



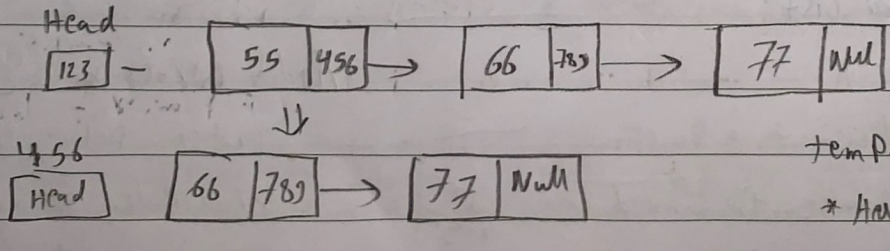
2- Void insert_at ~~From~~ _end (Node ** Head);



~~Insert~~ 3- Void insert After (Node * Head, uint32 Pos);



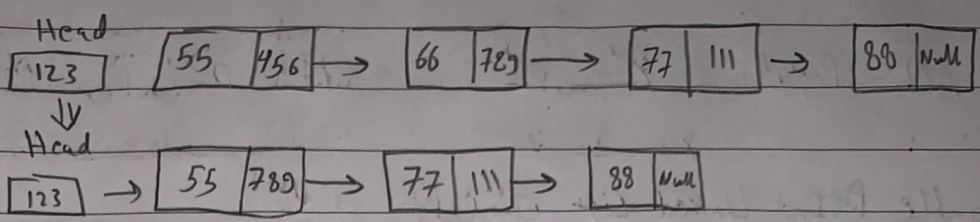
4- Void delete_beg (Node ** Head);

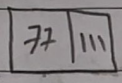


التاريخ: / / الموضوع: _____

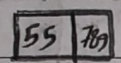
delete_specified_node(Node *Head);

(2)





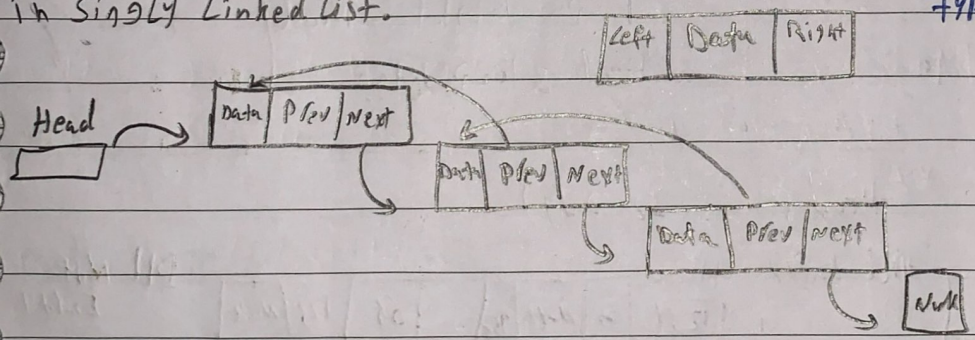
L After : L2

before L1 

L1->Link = &L2->Data;

Display
```
while(Null != temP){
    Printf(temP)
    temP= temP->link
```
Count
```
while(Null != temP){
    Count ++;
    temP= temP->link;}
```

Double Linked List : Contain an extra Pointer, typically called Previous Pointer, Left Pointer, together with next Pointer, Right Pointer and data which are there in Singly Linked list.




```
typedef struct Node{
    int Data;
    struct Node Left;
    // // // Right;
};
```

Advantage over single Linked List : A) a DLL can be transerved in both directions from forward or Backwards

B) the delete operation in DLL is more efficient if pointer to the node to be delete is given

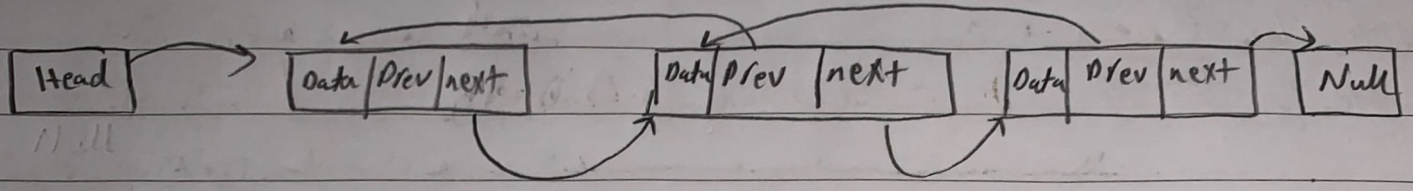
C) we can quickly insert a new node before a given node

التاريخ: / / الموضوع: _____

Disadvantages: A) every node of DLL Require extra space for an left Pointer

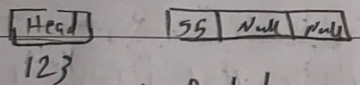
B) All operations require an extra Pointer Previous to be maintained

Double ~~Point~~ Linked List Concept;



Insert at beg; insert (Node ** Head, int32 data)

DLL Empty

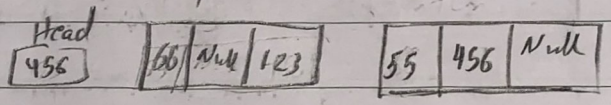


temP data

temP left = Null

temP right = Null

Head = &temPdata;



temP data ✓

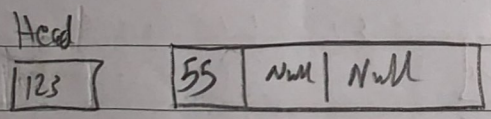
temP left = *Head;

temP Right = Null

Head = &temP data;

L1 left = &temP data

Insert at end; insert (Node ** Head, int32 data)



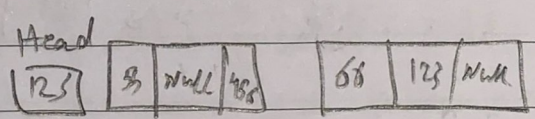
DLL Empty

temPdata

temPleft = Null

temPright = Null

Head = &temP Data



DLL Not Empty

temPdata ✓

temP Left = &L1 Data;

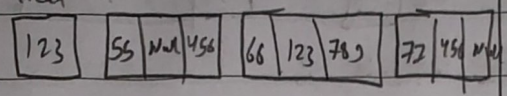
temP Right = Null;

التاريخ: / /

الموضوع: _____

insert After; insert (Node* Head, uint32 Pos, uint32 Data)

if it is the last node (if L1 Right = Null)



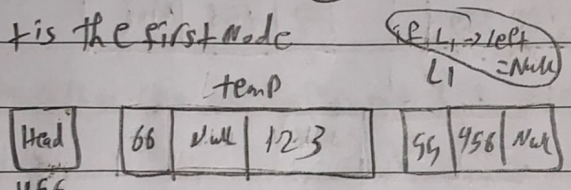
Pos (2)

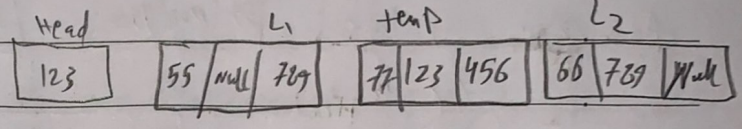
tempdata ↙
temp right = Null
temp left = &L1 Data
L1 -> Right = &temp data

tempdata ↙
temp right = &L2 Data;
temp Left = &L1 Data;
L1 -> Right = &temp data;
L2 -> Left = &temp data;

insert Before; insert (Node ** Head, uint32 Pos, uint32 Data)

if it is the first node (if L1 -> left = Null)

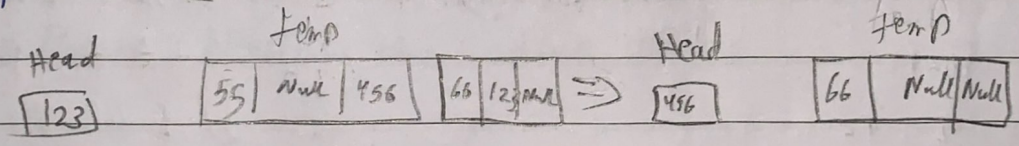




tempdata ↙
temp right = &L1 Data;
temp. Left = Null;
*Head = &temp data; / L1 -> Right = &temp data

tempdata ↙
temp right = &L2 Data;
temp Left = &L1 Data;
L1 Right = &temp Data;
L2 Left = &temp data;

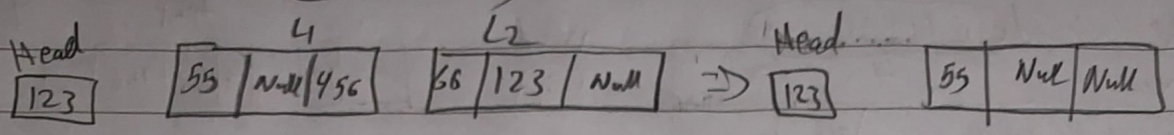
Delete beg; delete (Node ** Head)



temp = *Head
temp = temp Right;
temp left = Null
*Head = temp

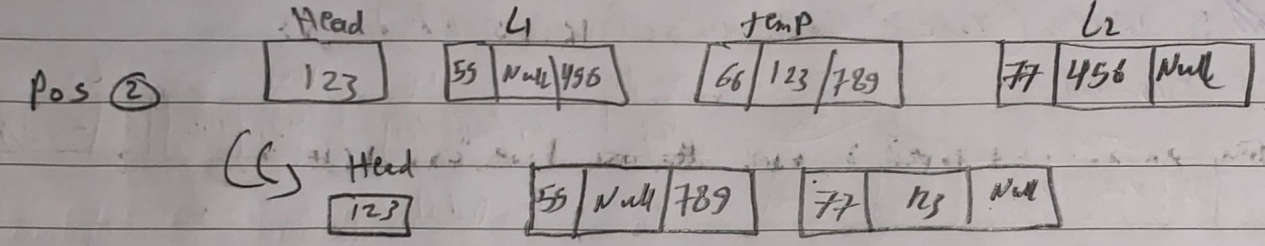
التاريخ: / / الموضوع: _____

Delete End: delete (Node *Head)



L2 -> Left = Null;
L1 -> Right = Null;
L2 = Null;
free (L2);

Delete Specified: delete (Node *Head, int32 Pos)



L1 -> Left = temp Left;
L2 -> Right = temp Right;
temp Left = Null;
temp Right = Null;
temp = Null;
free (temp)