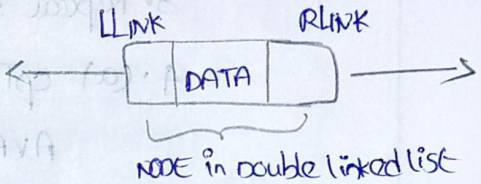
Double Linked List :—

→ In a single linked list one can move through the list of data only in one direction (Left TO Right) only.

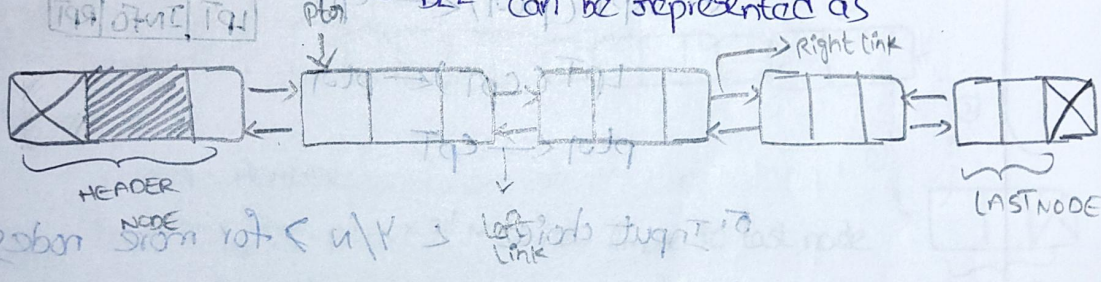
→ so the single linked list is also known as "one way list".

→ A double linked list is a two way list. That is we can traverse the nodes either from left to Right / Right to left (Both the directions)

→ Here the node is represented as



→ Structure of nodes in the DLL can be represented as



→ From the above diagram we observe that LLINK, RLINK points to left and Right side of the nodes.

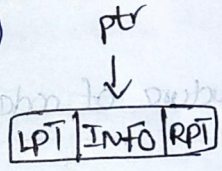
→ Every node Except HEADER and LASTNODE points to previous and successing nodes.

Operations on double linked list -:

→ creation
→ Traversing — 1) forward traversing
2) Backword traversing
→ Insertion operation
→ deletion operation
→ Searching

i) creation of double linked list :—

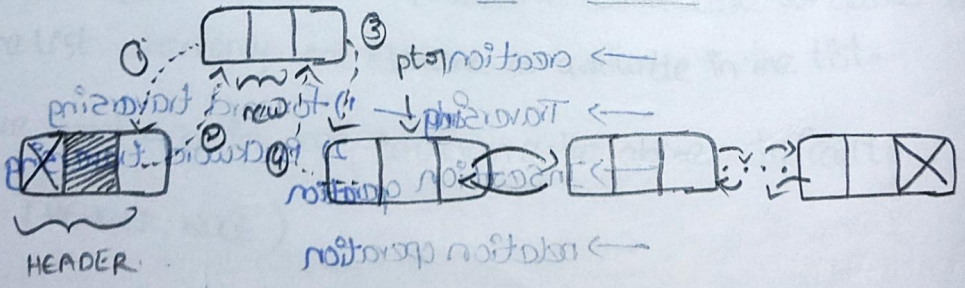
1. ptr ← AVAIL
   AVAIL ← RPT (AVAIL)
   READ INFO (ptr)
   LPT (ptr) ← NULL
   FIRST ← ptr
2. STORE "y" to CH
3. Repeat step 4 to 5 while CH = "y"
4. (a) cpT ← NULL
   AVAIL ← RPT (AVAIL)



   (b) RPT (ptr) ← cpT
   LPT (cpT) ← ptr
   ptr ← cpT
5. Input choice < Y/N > for more nodes
6. RPT (ptr) ← NULL
7. STOP

Insertion operation on double linked list :—

(i) Inserting a node at the front



1. ptr = HEADER.RLINK //points to the first node
2. new = GETNODE(NODE) //Avail a new node from memory Bank
3. If (new ≠ null) then

1. new.LLINK = HEADER

2. HEADER.RLINK = new

3. new.RLINK = ptr

4. ptr.LLINK = new

5. new.DATA = X

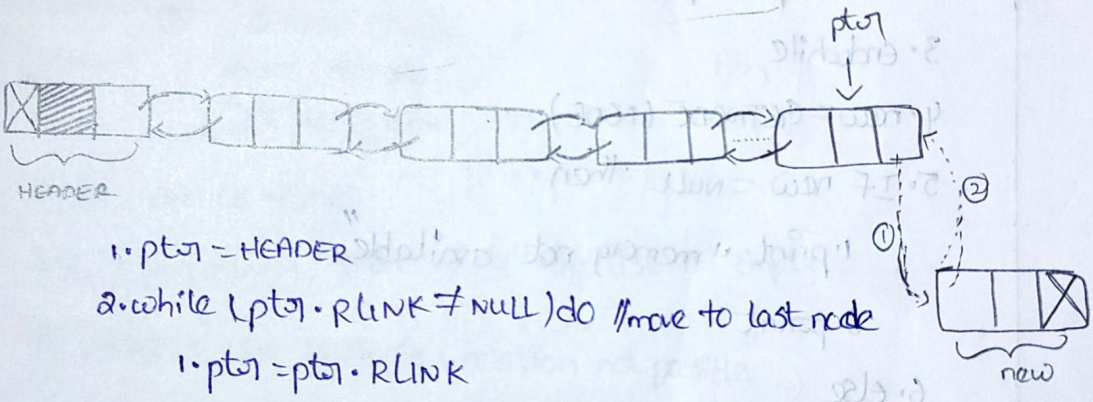
4. else

1. print "unable to allocate memory : insertion not possible."

5. End if

6. STOP

(ii) Inserting a node at the End



1. ptr = HEADER

2. while (ptr.RLINK ≠ NULL) do //move to last node

1. ptr = ptr.RLINK

3. End while

4. new = GETNODE(NODE) //Avail a new node from memory bank

5. If (new ≠ NULL) then // If node is available in memory bank

i) new.LLINK = ptr

ii) ptr.RLINK = new

iii) new.RLINK = NULL

iv) new.DATA = X

6. Else

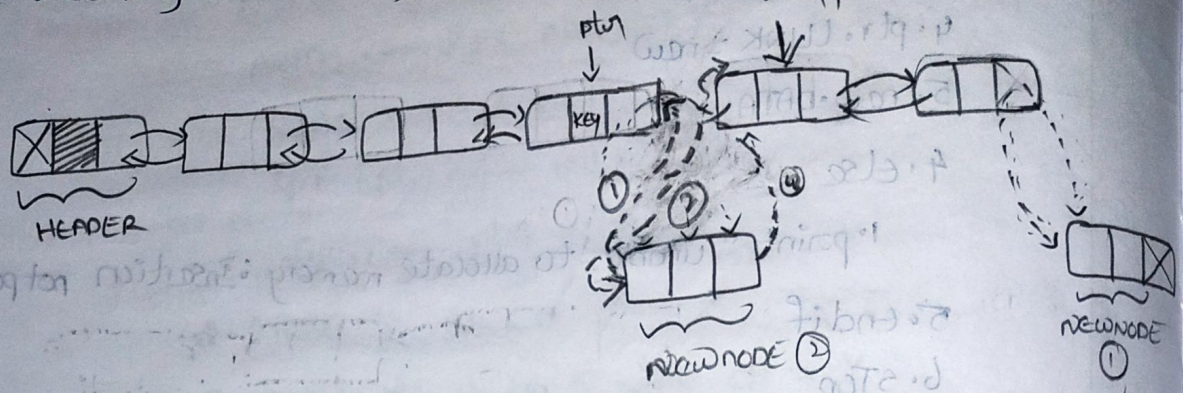
print "unable to allocate memory : so Insertion not possible."

7. End if

8. STOP

(ii) ~~(Inserting operation)~~

(iii) Inserting a node at any Intermediate position



1. ptr = HEADER
2. while (ptr·DATA ≠ KEY) and (ptr · RLINK ≠ NULL) do
   1. ptr = ptr · RLINK
3. endwhile
4. new = GETNODE (NODE)
5. If new = NULL then
   1. print "memory not available"
   2. print
6. else
   If (ptr · RLINK = NULL) then
   ptr · RLINK = new
   new · LLINK = ptr
   new · RLINK = NULL
   new · DATA = X
   } Insert at End of the list
7. else
   1. ptr1 = ptr · RLINK
   2. new · LLINK = ptr
   3. ptr1 · LINK = new
   4. ptr · RLINK = new
   5. new · RLINK = ptr1
   6. ptr = new
   } Insert at key position

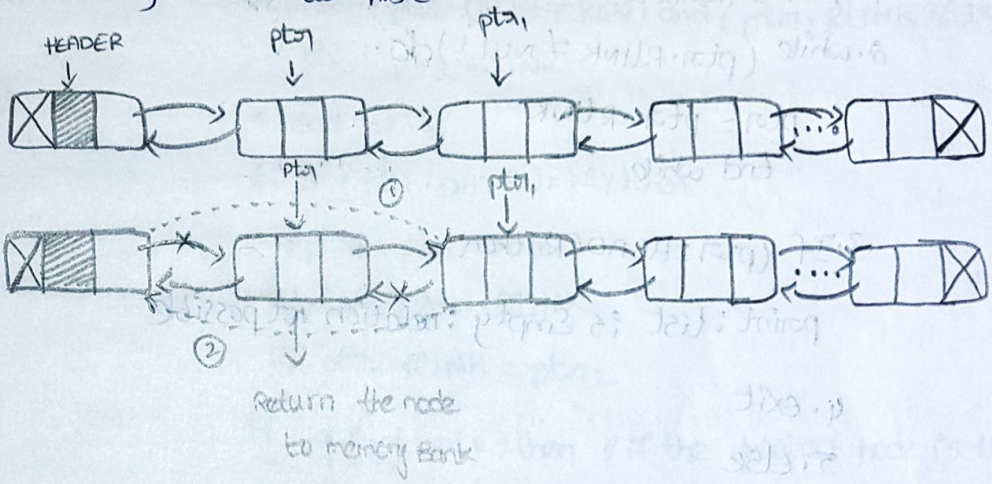
7. new.DATA = X

8. End if

9. STOP

Deleting a node in double linked list :—

i) Deleting a node at first



1. ptr = HEADER.RLINK

2. If (ptr = NULL) then // If the list is Empty

(i) point : "List is Empty : deletion not possible"

(ii) Exit

3. Else

i) ptr1 = ptr.RLINK // points to 2nd node

ii) HEADER.RLINK = ptr1

iii) If (ptr1 ≠ NULL)

ptr1.LLINK = HEADER

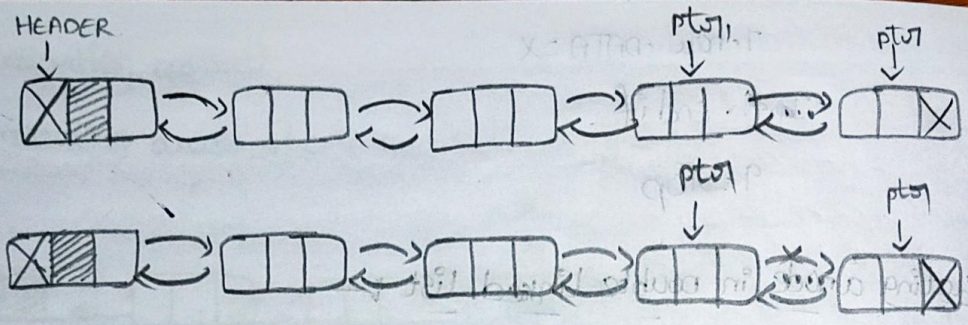
End if

4. RETURNNODE (ptr)

5. End if

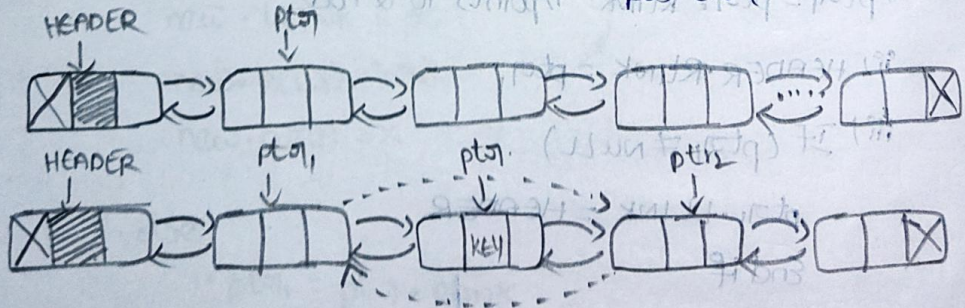
6. STOP

ii) Deleting a node at End.



1. ptr = HEADER
2. while (ptr.RLINK ≠ NULL) do
   ptr = ptr.RLINK
   End while
3. If (ptr = HEADER) then
   print : List is Empty : deletion not possible
4. exit
5. else
   i) ptr1 = ptr.LLINK
   ii) ptr1.RLINK = NULL
   (iii) RETURNNODE (ptr)
6. end if
7. STOP

iii) Deleting a node at Intermediate position



Element to be deleted and Return to memory Bank

1. ptr_1 = HEADER.RLINK

2. while (ptr_1.RLINK ≠ NULL) do

i) print visit is Empty deletion not possible

ii) exit

3) End if

4) while (ptr_1.DATA ≠ KEY) and (ptr_1.RLINK ≠ NULL) do

ptr_1 = ptr_1.RLINK

5) End while

6) If (ptr_1.DATA) = KEY) then

i) ptr_{11} = ptr_1.LLINK

ii) ptr_{12} = ptr_1.RLINK

iii) ptr_{11}.RLINK = ptr_{12}

If (ptr_{12} ≠ NULL) then // if the deleted node is the last node.

4) ptr_{12}.LLINK - ptr_{11}

5) End if

6) RETURN NODE (ptr_1)

7) Else

print : The node doesnot exist in the given list.

8) End if

9) STOP.

Polynomial representation Linked List

Addition of two polynomials (Java program & algorithm)

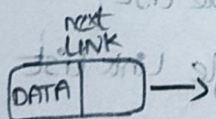
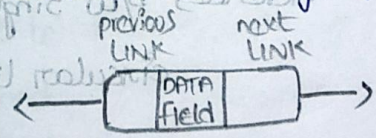
Differences b/w Arrays & Linked List

Differences b/w single Link & Double List

Circular link list & single Link List

Array	Linked List
1. Array is a collection of variables of some datatype.	1. A linked list is a set of nodes each node has two fields is called as linked list.
2. The data is store in contigous memory location.	2. In linked list we store the elements randomly (or) any where in the memory zone.
3. The size of memory is fixed and it is not possible to change during runtime.	3. The memory allocation in the linked list can be done in dynamically at the runtime.
4. The elements are not depended Each other.	4. Elements are dependent Each other.
5. The memory is allocated in not Effective.	5. The memory is allocated at run time.
6. utilization of memory at the compilation time.	6. The utilization of memory is very effect.
7. Accessing of Elements is faster when compared linked list.	7. Accessing of Elements take more time then arays.

Differences Between single linked list and double linked list

1) A single linked list is collection of nodes where Each node contains two parts data and link. 	1. A double linked list is collection of nodes where Each node contains 1 datafield left & right pointers. 
2) The Elements can be access using next link.	2. The Elements can be access using both previous link and next link.

3. no extra field is require to access the nodes in the list so the nodes take less amount of space.	3. Extra field is required to store previous link. so the node in double linked list it occupy more amount of space in the memory.
4. In SLL traversal of the data is done the only in one direction. so SLL is also known as one way listing.	4. In DLL traversal of data is done in two ways (forward and backward) so DLL is also known as two way listing.
5. single linked list are generally used for implementation of STACKS.	5. DLL is generally used to implement HEAP concept and BINARY TREES.

STACKS :—