Data structures :

A data structure is a specialized format for organizing processing, retrieving and storing data.

Cases in Algorithm :-

i) worst case

ii) Average case

iii) Best case

Methods of Algorithm :-

1. Greedy method
2. Divide method
3. Dynamic programming
4. Back tracking approach
5. Branch & Bound

Array :- An array is a finite collection of homogeneous data Elements.

* It is a finite it stores limited no.of Elements.
* Elements are stored in the contiguous memory locations.
* Data to be stored in the form of same type/ homogeneous

Based on the index, arrays are classified into two types.

1. Single dimensional Array
2. multi-dimensional Array

operations on Arrays :-

1. Traversal (visit all the Elements in array one time)
2. Inserting (Adding Elements to array)

3. delate (delate an Element in array)

4. sorting (Arranging Elements in sequencing manner i.e. asce, desc

5. searching

operations on Array :-

1) Traverse / Traversing :-

i) Elements in an Array.

ii) Address / Index of 1st Element (L)

(L: lower bound value)

iii) index of nth Element (u)

(u: upper bound value)

iv) PROCESS ( )

1. i=L

2. while i≤u

3. PROESS (A[i])

4. i=i+1

5. END while

6. stop

Traverse

1. Elements in array
2. Address / Index of 1st E
3. In nth E
4. process ( )
i=L
while i≤u
process A[i]
i=i+1
Endwhile
stop

2) Searching :- → Element to be searched

1. L=0, KEY value, u="max no of Elements in array";

Location=0

2. while (i≤u and found = = 0) do

If COMPARE (A[i], KEY) = TRUE then

i) found = 1

ii) Location = 1

Else

i = i+1

3. END

4. END while

5. If (found = 0) then

print "Search unsuccessful"

"KEY" Element not found in the list

Else

print " Search is successful

"Element found at location : location"

7. END if

8. Return (location)

9. stop

Insertion operation in Array :-

(1) Insertion of Element from the Array

(2) Deletion of Element from the Array

(1) Insertion of Element from the Array

(i) At the End,

(ii) At the Beginning

(iii) At any position / specific

(i)

0	1	2	3	4	5	6	7	8	9
1	2	3	10						

A[i] = u = MAX

overflow of List : Insertion not possible

STOP

READ DATA

UB ← UB+1

A [UB] ← DATA

STOP

ii) K ← UB

Repeat step (5) until K ≥ LB

A [K+1] ← A [K]

K ← K-1

A [LB] ← DATA

iii) Insertion at any position in an array

i) If UB = max then

write " Array over flow : Insertion not possible

STOP

ii) READ DATA

READ LOCATION

K ← UB

Repeat the step 5 until K ≥ LOC

A [K+1] ← A [K]

K ← K-1

A [LOC] ← DATA

STOP

Ex-: LOC = 4 UB = 6 DATA = 10

K ≥ LOC

K = 6

6-1 = 5

0	1	2	3	4	5	6	7
1	3	5	7	8	9		11

0	1	2	3	4	5	6		
1	3	5	7	10	8	9	11	

Daleting :—

i) Dalate an Element from End of the list.

ii) Dalate an Element from begining of the list.

iii) Dalate an element at given position

i) N = (UB − LB) + 1

If N = 0

List is underflow : deletion not possible

STOP

A [UB] ← NULL Max = 10

UB ← UB − 1

STOP

Linked List-:

LINK

DATA	•

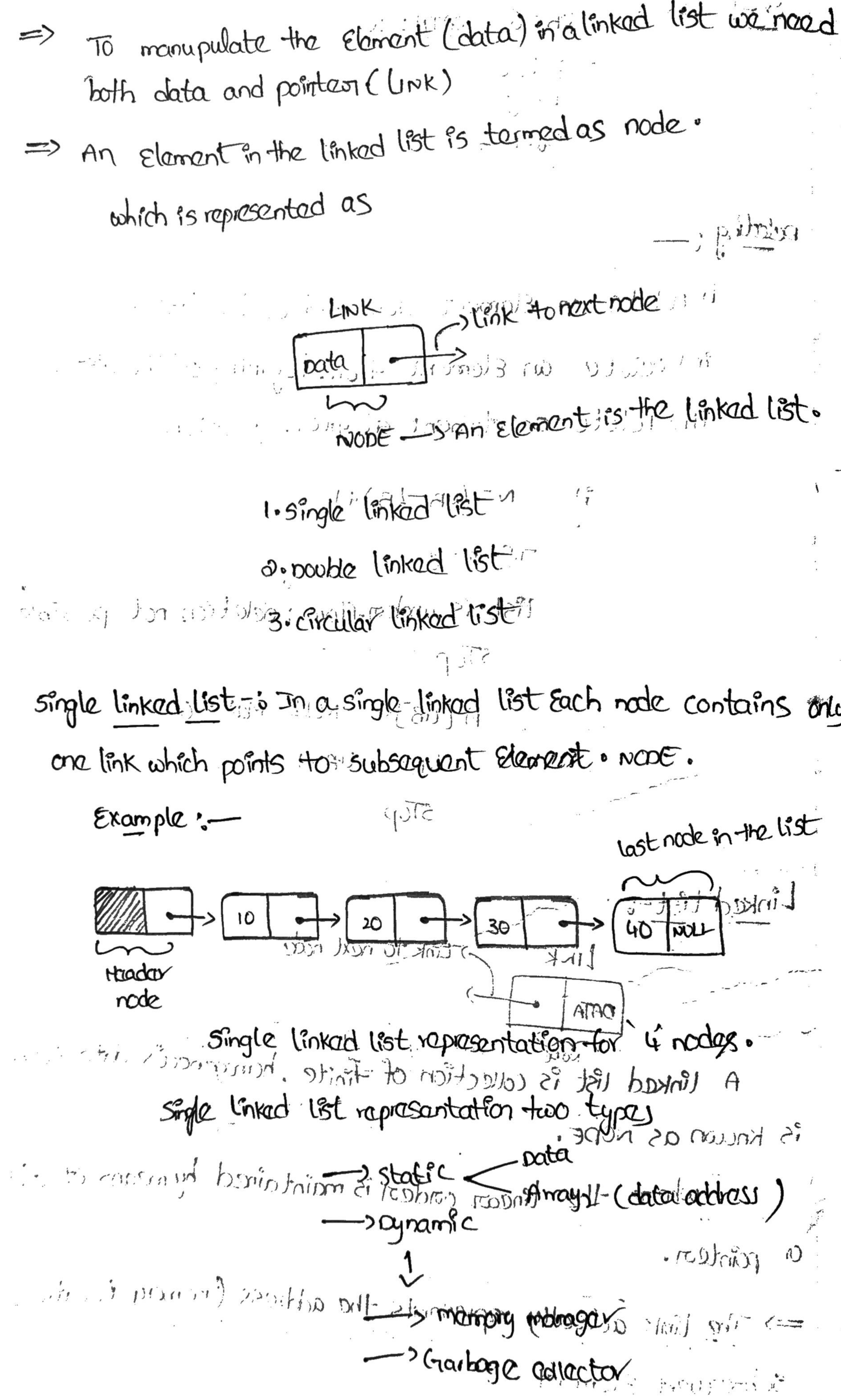
→ Link to next node

NODE

A linked list is collection of finite, homogeneous data Elements is known as NODE.

(where) the linear order is maintained by means of a LINK or pointer.

⇒ The link actually represents the address (memory Location) of subsequent Element.



⇒ To manupulate the Element (data) in a linked list we need both data and pointer (LINK)

⇒ An Element in the linked list is termed as node.

which is represented as

1. single linked list

2. Double linked list

3. circular linked list

Single linked list :- In a single linked list Each node contains only one link which points to subsequent Element. NODE.

Example :-

Single linked list representation for 4 nodes.

Single linked list representation two types

→ static < Data / Array 11 - (data address)

→ Dynamic

↓

→ memory manager

→ Garbage collector

Operations on linked list:-

- → Traverse
- → Insertion
- → deletion
- → sorting
- → searching
- → Merging

Traversing:- Traversing means visiting every node in the list starting from first node to the last node.

Algorithm steps:-

```
ptr = HEADER.LINK
while (ptr ≠ NULL) do
  PROCESS (ptr)
ptr = ptr.LINK
End while
Stop.
```

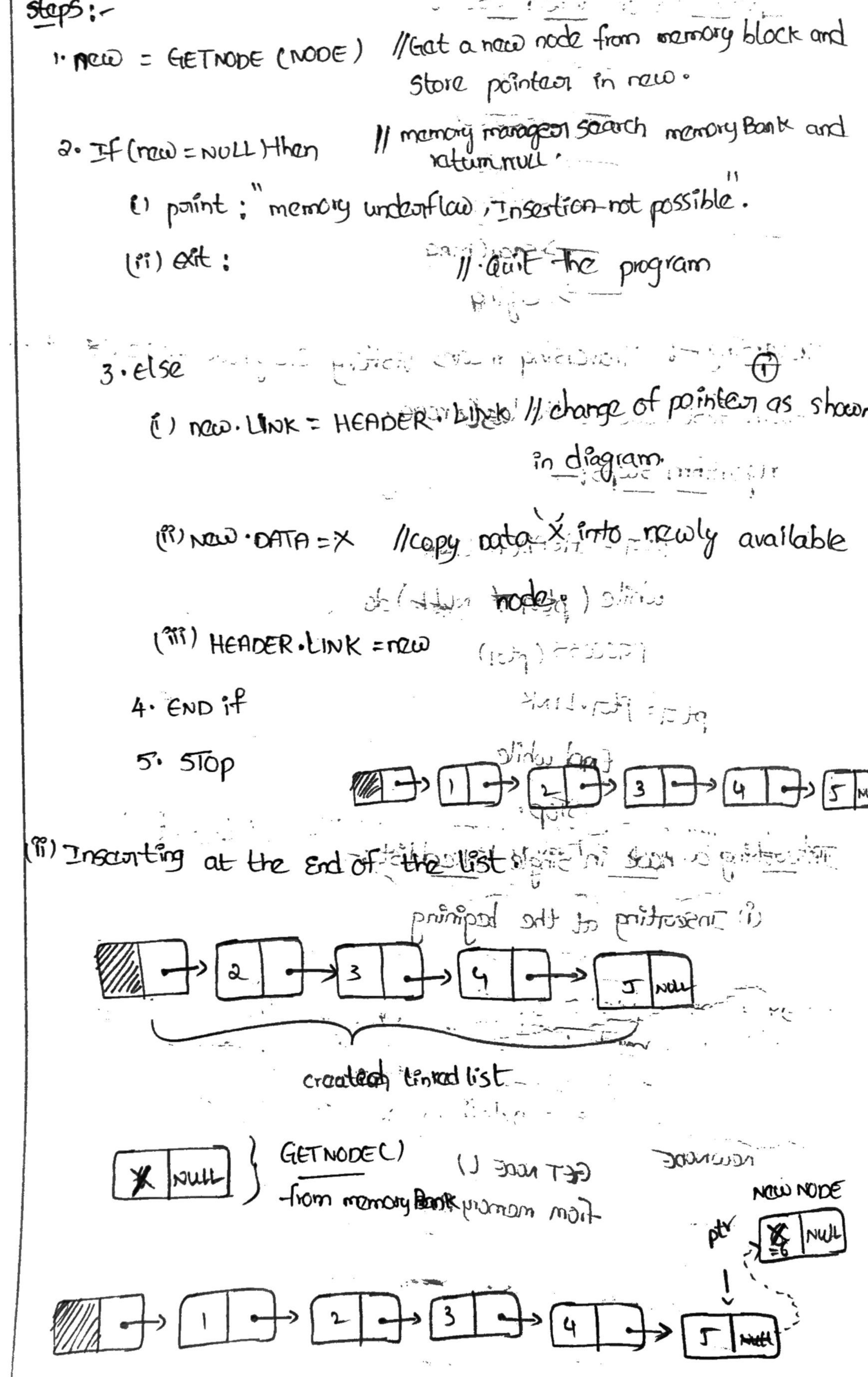
Inserting a node in single linked list:-

(i) Inserting at the begining

available created linked list

NEWNODE

GET NODE ()
from memory



Steps :-

1. new = GETNODE (NODE) //Get a new node from memory block and store pointer in new.

2. If (new = NULL) then // memory manager search memory Bank and return null.

 (i) print : "memory underflow, Insertion not possible".

 (ii) exit : // Quit the program

3. else

 (i) new.LINK = HEADER.LINK // change of pointer as shown in diagram

 (ii) NEW.DATA = X //copy data X into newly available node.

 (iii) HEADER.LINK = new

4. END if

5. STOP

(ii) Inserting at the End of the list

created linked list

GETNODE() from memory Bank

NEW NODE

ptr

X NULL

steps :—

1. new = GETNODE (NODE) // Get a new node from memory block and store pointer in new

2. If (new = NULL) then // memory manager search memory Bank and returns NULL.

i) print : "memory under-flow Insertion not possible"

ii) Exit // Quit the program

3. Else

i) ptr = HEADER

(ii) while (ptr.LINK ≠ NULL) do

i) ptr = ptr.LINK

(iii) End while

(iv) ptr.LINK = new

(v) new.DATA = X = 6

4. Endif

5. Stop

[header] → [1|•] → [2|•] → [3|•] → [4|•] → [5|•] → [6|NULL]

(iii) Inserting at key position in the list

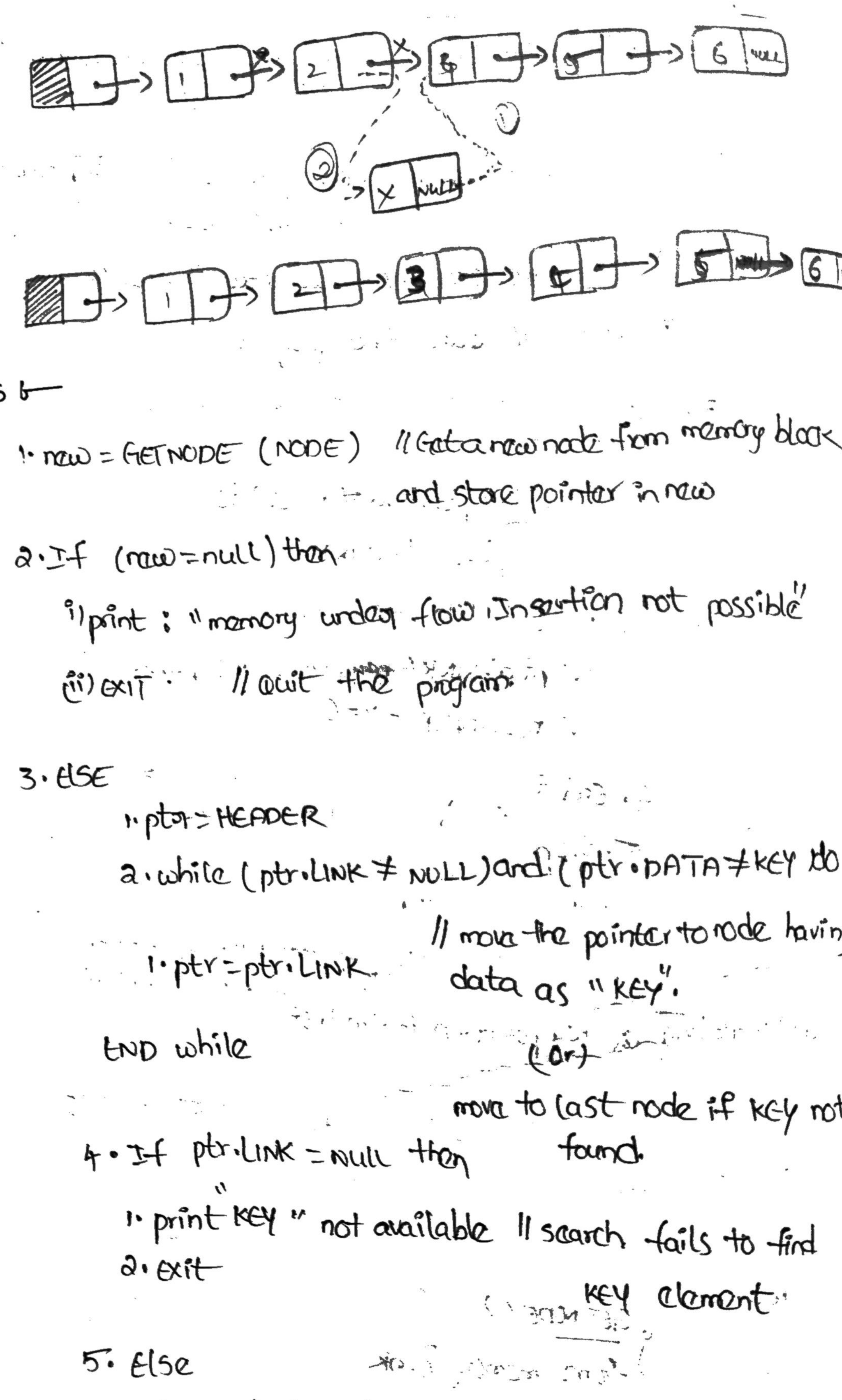
[header] → [1|•] → [2|•] → [4|•] → [5|•] → [6|NULL]

created linked list

NEW NODE

[X|NULL] } GET NODE ( ) from memory Bank

Fig



Steps :-

1. new = GETNODE (NODE) // Get a new node from memory block and store pointer in new

2. If (new = null) then
   i) print : "memory underflow, Insertion not possible"
   (ii) EXIT // quit the program

3. ELSE
   1. ptr = HEADER
   2. while (ptr.LINK ≠ NULL) and (ptr.DATA ≠ KEY) do
      1. ptr = ptr.LINK // move the pointer to node having data as "KEY".
   END while
   (or)
   move to last node if KEY not found.

4. If ptr.LINK = null then
   1. print "KEY" not available // search fails to find KEY element
   2. exit

5. Else
   1) new.LINK = ptr.LINK
   2) ptr.LINK = new

5. new.DATA = X = 3

6. End if

7. End if

8. STOP

Deletion of a node in a single linked list :-

i) Deleting a node at the begining/front of the list
ii) Deleting a node at end of the list
iii) Delete a node at required position

} Here we use RETURNNODE ( )

i) Deleting a node at front of the list

ptr

[ ] → [5] → [1] → [2] → [3] → [4|NULL]

ptr1

ptr

[ ] → [5] →x [1] → [2] → [3] → [4|NULL]

key ptr1

[ ] → [1] → [2] → [3] → [4|NULL]

final list after deleting element

1. ptr = HEADER.LINK

2. If (ptr = NULL)   // If list is empty

 1. print list is Empty

 2. exit

3. Else

 1. ptr1 = ptr.LINK   // ptr1 points to next node

 2. HEADER.LINK = ptr1   // next node becomes first node

3. RETURNNODE (ptr) //delete node moved to memory bank for the further use.

4. ENDif

5. STOP

1. ptr = HEADER.LINK

2. If (ptr = NULL) then

 1. print "The List is Empty : NO Deletion"

 2. Exit

 Else

 1. ptr1 = ptr.LINK

 2. HEADER.LINK = ptr1

 3. Return NODE (ptr)

 4. ENDIf

 5. STOP

5 | 1 | 2 | 3 | NULL

ptr1 | ptr | key element

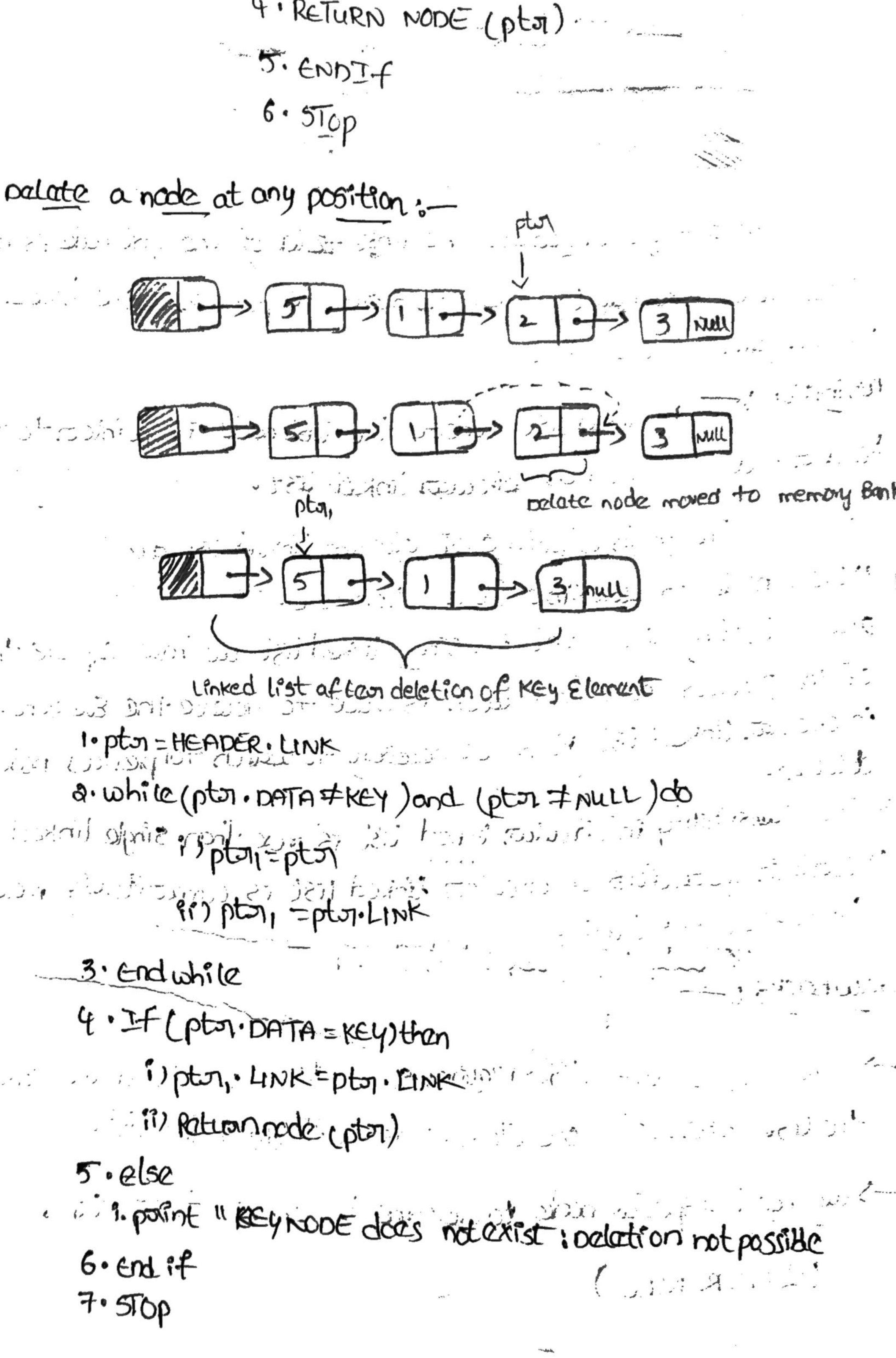
1. while (ptr ≠ NULL) do

 ptr1 = ptr

 ptr = ptr.LINK

2. End while

3. ptr1.LINK = NULL



3. ptr1.LINK = null

4. RETURN NODE (ptr)

5. ENDIf

6. STOP

delate a node at any position :-

1. ptr = HEADER.LINK

2. while (ptr.DATA ≠ KEY) and (ptr ≠ NULL) do

   i) ptr1 = ptr

   ii) ptr1 = ptr.LINK

3. End while

4. If (ptr.DATA = KEY) then

   i) ptr1.LINK = ptr.LINK

   ii) Return node (ptr)

5. else

   1. print "KEY NODE does not exist : delation not possible

6. End if

7. STOP

Circular linked list :—

null } normal list

} circular linked list

In a single linkedlist the link field of the last node is null. If we utilize the null link field to store the pointer to the header node. we can gain an advantage.

Definition :— A linked list where the last node is pointer to the header node is called as circular linked list.

The main advantage of circular linked list are

1) Every node is accessible from a given node.
2) why deleting a Element in a single linked list we have capture the address of the previous node and search is made to delete the Everyone. But in circular linked list it is not necessary to search for previous node while deletion.
3) The accessibility in circular linked list is more than single linked list.
4) certain operations or circular linked list is comparitively more Effective than single linked list.

DRAW BACKS :—

→ The circular linked list requires some Extra care to detect End of the list when only one Element is available in the list.

→ we need a special node to overcome the above difficulty. (HEADER NODE)

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

LLINK DATA RLINK

NODE in double linked list

-> Structure of nodes in the DLL can be represented as

ptr

Right link

HEADER NODE

left Link

LAST node

-> 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) Backward traversing

-> Insertion operation

-> deletion operation

-> Searching

1) 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

ptr ↓ | LPT | INFO | RPT |

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

new

HEADER

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. endif

6. STOP

(ii) Inserting a node at the end

HEADER ptr new ① ②

1. ptr = HEADER
2. while (ptr.RLINK ≠ NULL) do // move to last node
   1. ptr = ptr.RLINK
3. endwhile
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. Endif
   8. STOP

(iii) ~~(Inserting operat)~~

(iii) Inserting a node at any Intermediate position

ptr ptr1

key

HEADER

NEWNODE (2) NEWNODE (1)

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. point
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. ptr_1 = ptr.RLINK
   2. new.LLINK = ptr
   3. ptr_1.LLINK = new
   4. ptr.RLINK = new
   5. new.RLINK = ptr_1
   6. ptr = new

   } Insert at key position

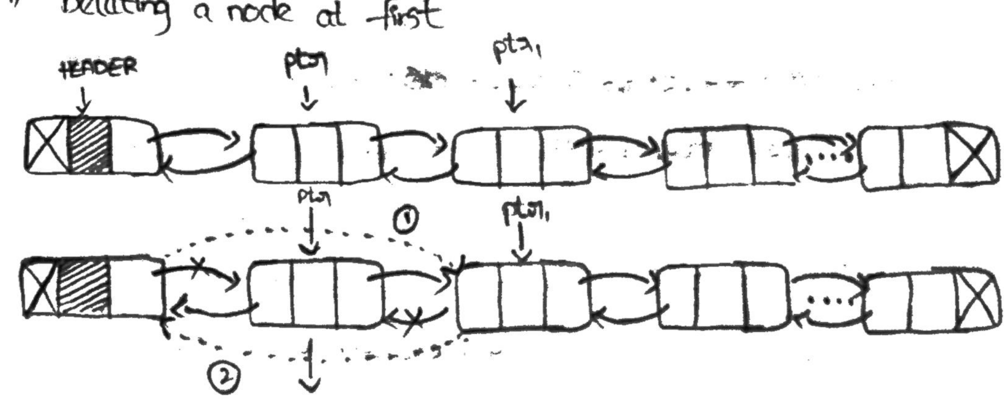
7. new . DATA = X

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) print : " 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

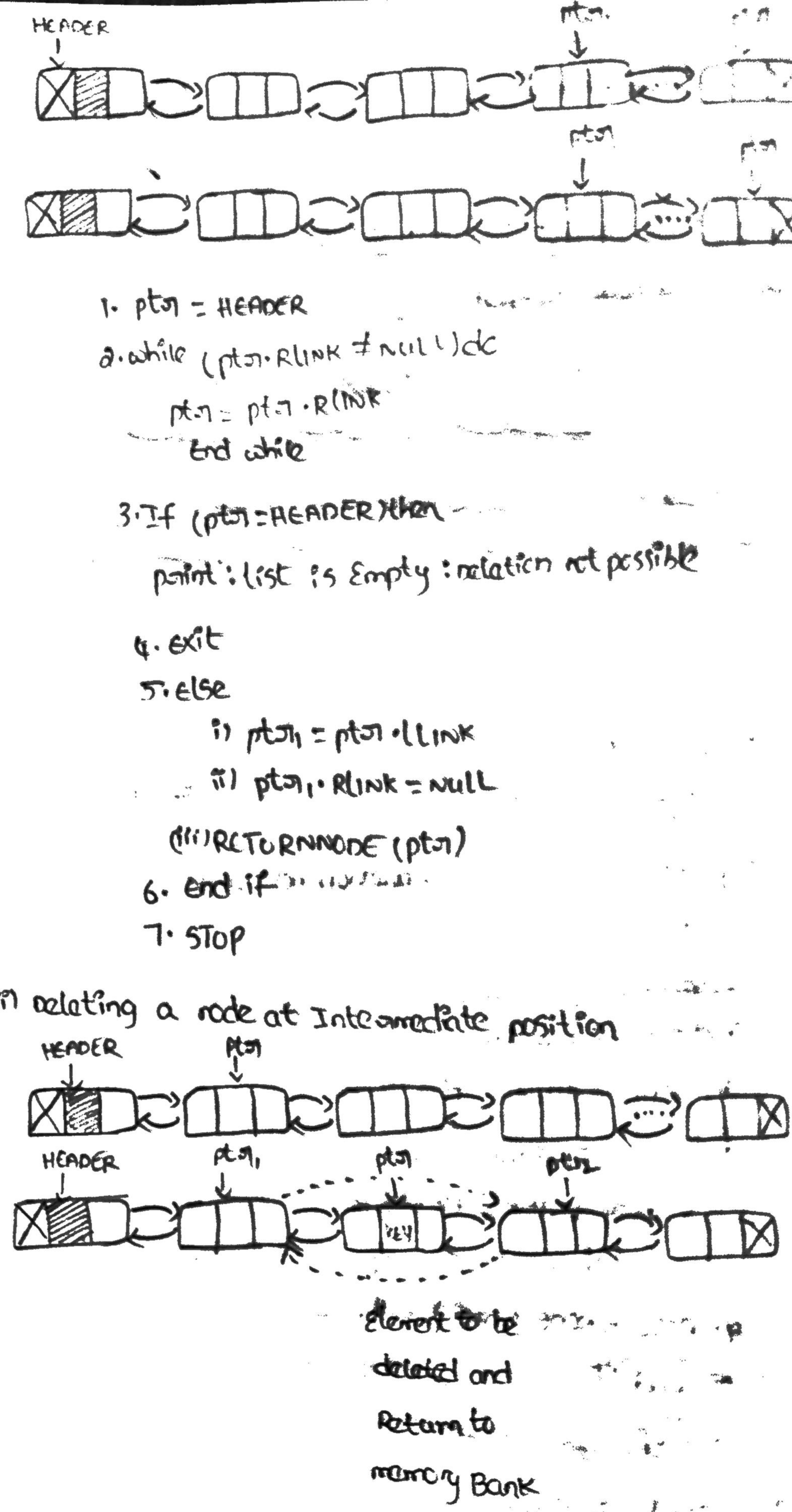
End if

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

1. ptr = HEADER.RLINK

2. while (ptr.RLINK ≠ NULL) do

i) print visit is Empty deletion not possible

ii) exit

3) End if

4) while (ptr.DATA ≠ KEY) and (ptr.RLINK ≠ NULL) do
ptr = ptr1.RLINK

5) End while

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

i) ptr_1 = ptr.LLINK

ii) ptr_2 = ptr.RLINK

iii) ptr_1.RLINK = ptr_2

If (ptr_2 ≠ NULL) then // if the deleted node is the last node.

4) ptr_2.LLINK - ptr_1

5) End if

6) RETURN NODE (ptr)

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 same 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 than arrays.

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.
next LINK DATA | →	previous LINK, next LINK ← | DATA field | →
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 :—

STACK Representation

→ A STACK is a collection of homogenous data Element where Insertion (PUSH) and deletion (pop) takes place only at one End called TOP.

→ An Element in a STACK is termed as "ITEM".

→ The maximum number of Elements that a STACK can hold is called "SIZE" of the STACK.