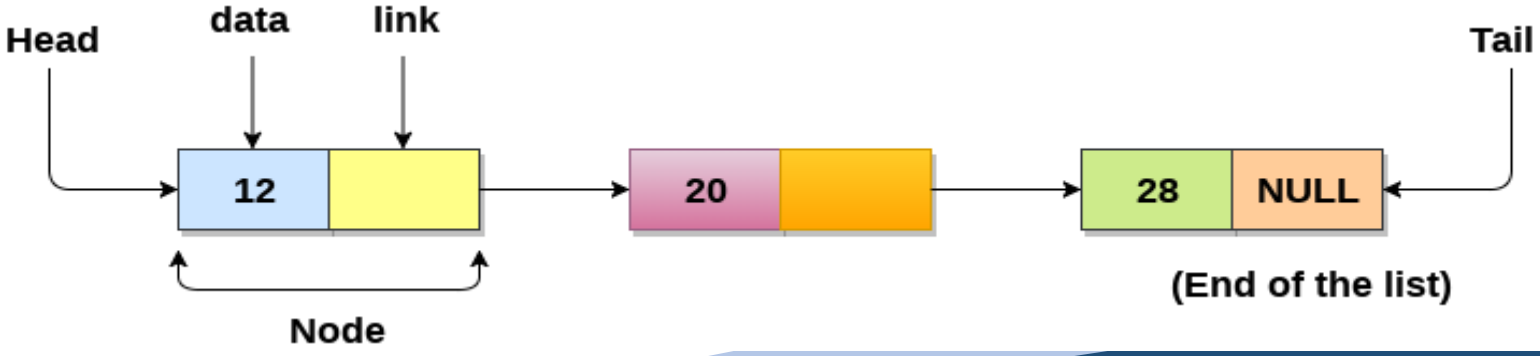
UNIT- FIVE
LINKED LIST

Topics to be Covered

- Introduction
- Linked list as an ADT
- Dynamic Implementation
- Insertion and deletion of Node To and From a list
- Insertion and deletion After and Before Nodes
- Linked stacks and Queues
- Doubly Linked list and its advantages

Introduction: Linked List

- A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers.
- Linked List can be defined as collection of objects called **nodes** that are randomly stored in the memory.
- A node contains two fields i.e. data stored at that particular address and the pointer which contains the address of the next node in the memory.
- The last node of the list contains pointer to the null.
- It is also called dynamic data structure.



Disadvantage of List (Array)

- The size of array must be known in advance before using it in the program.
- Increasing size of the array is a time taking process. It is almost impossible to expand the size of the array at run time.
- All the elements in the array need to be contiguously stored in the memory. Inserting any element in the array needs shifting of all its predecessors.

Why Linked List? (Advantages)

- It allocates the memory dynamically. All the nodes of linked list are non-contiguously stored in the memory, instead they are linked together with the help of pointers.
- Sizing is no longer a problem since we do not need to define its size at the time of declaration.
- List grows as per the program's demand and limited to the available memory space.
- Insertion and deletion of nodes are easier and efficient
- Complex applications can be easily carried out with linked list

Disadvantages of Linked List

- More memory is required to handle linked list.
- Accessing to arbitrary data is a bit time consuming.

Categories of Linked List

- Linear Linked List
- Doubly Linked List
- Circular Linked List
- Header Linked List
- Circular doubly linked list

- Linear Linked List: Each node is divided in two parts: First part contains the information and Second part contains the address of the next node in the list. Each node has a single pointer to the next node and the last node points to NULL pointer.

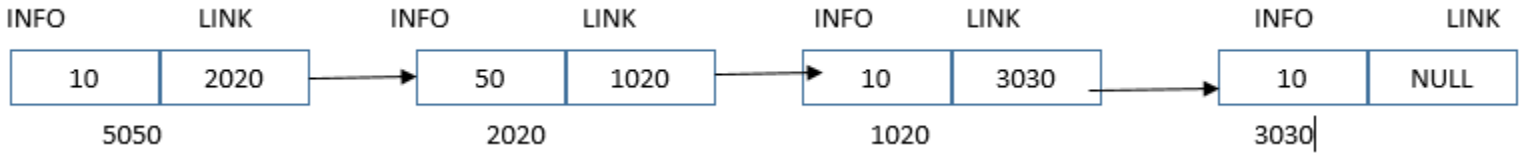
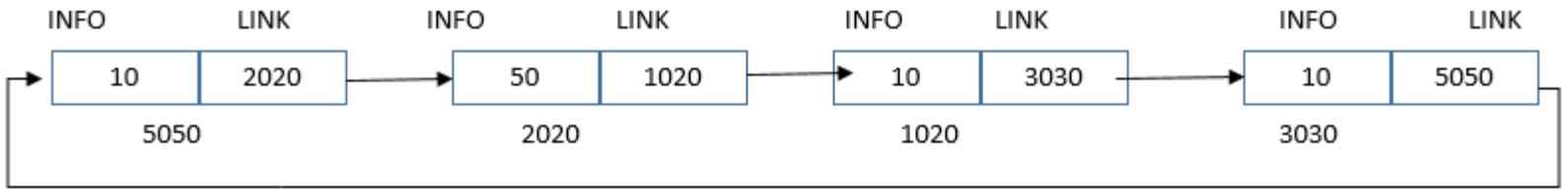


Figure: Logical representation of Linear linked List

- Circular Linked list: Similar to linear pointer except that the second part of the last node points to the address of first node. It doesn't contain the NULL pointer



- Doubly Linked List: Also called two- way Lists. In case of doubly linked list, two link fields are maintained instead of one as in singly linked list which helps in accessing both successor and predecessor nodes.

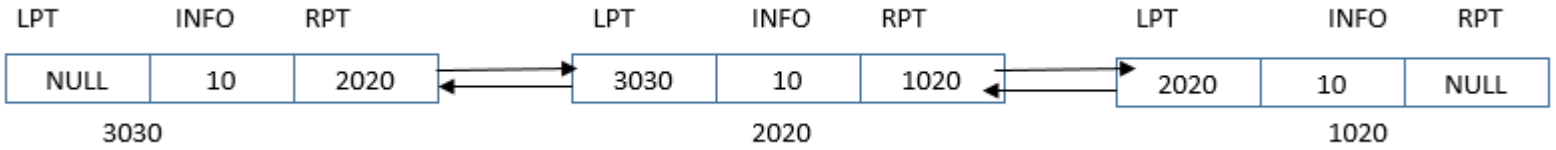
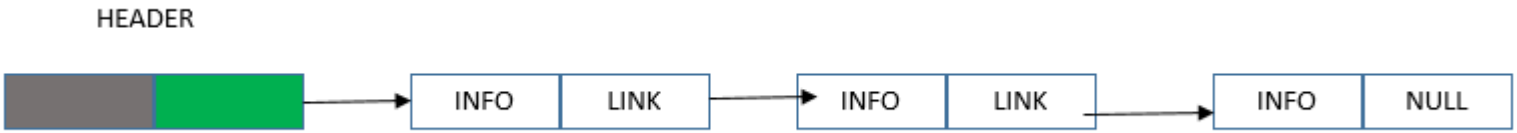


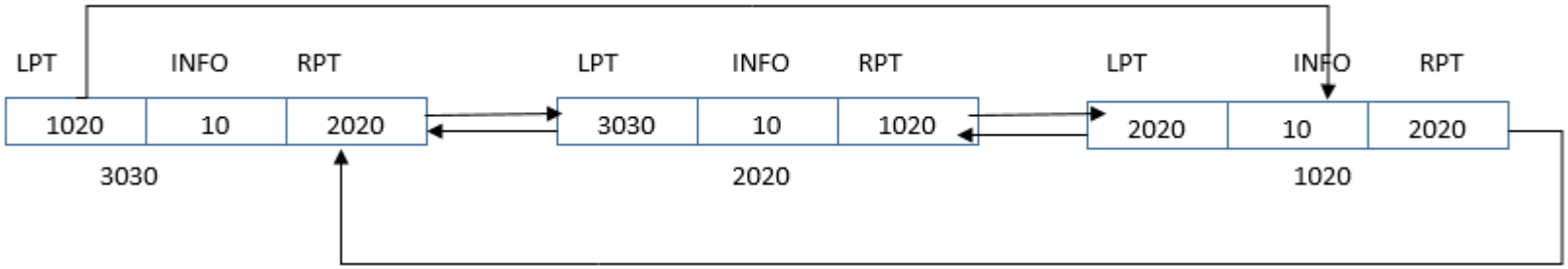
Figure: Logical representation of Doubly linked List

- Header Linked list: It always contains a single node, called a header node at the beginning of the linked list. The header node contains vital information about the linked list. Eg: number of nodes in the lists, sorted list or not etc.
- The data part of this node is generally used to hold any global information about the entire linked list. The next part of the header node points to the first node in the list.



- Circular Doubly Linked list: It has both successor and predecessor pointer in circular manner. It simplifies the insertion and deletion operations performed on doubly linked list

  - In Circular doubly linked list, the LPT of first node contains the address of last node and RPT of the last node contains the address of the first node.



Singly Linked List

- Is also called one way list.
- Each node is divided into two parts: INFO and LINK
- INFO contains information of the element
- LINK contains the address of next node
- The list with no nodes is called empty list or the null list and is denoted by the null pointer in the list pointer variable FIRST. i.e, FIRST=NULL

Basic Operations

1. Insertion:
   - At the end of LL
   - At the beginning of the LL
   - At after a given information node
   - In the sorted Information List
2. Deletion
   - From the end
   - From the beginning
   - information of the given node
3. Traversing a Singly Linked List
4. Sorting the Linked List
5. Searching a node in Linked list
6. Concatenation of Two Linked List
7. Reversing a list

- Linked List creation in C:

```
struct node
{
        int info;
        struct node *link;
};
struct node *FIRST;
```

- Allocation of memory dynamically:

```
PTR= (struct node*) malloc (size of (struct node))
```

- Accessing the elements of Structure using pointer:

```
PTR→info                or (*PTR).info
PTR →link               or (*PTR).link
```

- Free the memory:

use function **free(PTR);**

Creation of Singly Linked List

Here PTR is a pointer variable of type Struct node which allocates is size dynamically.
PTR= (struct node*) malloc (size of (struct node))

Algorithm:

1. Start
2. PTR= AVAIL
   AVAIL=LINK(AVAIL)
   READ INFO(PTR)
   FIRST =PTR
3. Repeat step 4 and 5 until the user choice “Y”
4. CPT=AVAIL
   AVAIL=LINK(AVAIL)
   READ INFO(CPT)
   LINK(PTR)=CPT
   PTR=CPT
5. Read user choice (Y/N) for more node
6. LINK(PTR)=NULL
7. Stop

Traversing of a singly linked list

Algorithm:

Step 1: Start

Step 2: PTR=FIRST

Step 3: REPEAT STEP 4 AND 5 WHILE PTR IS ! NULL

Step 4: PRINT INFO(PTR)

Step 5: PTR=LINK(PTR)

Step 6: Stop

Insertion in a Singly Linked List

Insertion is possible at following positions:

a. At the beginning
b. At the end
c. At after given information node
d. In the sorted information list

Note: before insertion, we have to check the **overflow** condition

If AVAIL==NULL, Overflow and stop

To create a new node:

PTR=AVAIL

AVAIL= Link(AVAIL)

Insertion at the beginning

Algorithm

Step 1: Start

Step 2: IF AVAIL=NULL then PRINT OVERFLOW AND STOP

Step 3: PTR=AVAIL

AVAIL= LINK(AVAIL)

READ INFO(PTR)

Step 4: LINK(PTR)=FIRST

FIRST =PTR

Step 5: STOP

Insertion at the end

Step 1: Start

Step 2: IF AVAIL=NULL then PRINT OVERFLOW AND STOP

Step 3: PTR=AVAIL

AVAIL= LINK(AVAIL)

READ INFO(PTR)

Step 4: CPT=FIRST

Step 5: Repeat Step 6 while Link(CPT)!=NULL

Step 6: CPT=Link(CPT)

Step 7: LINK(CPT)=PTR

LINK(PTR) =NULL

Step 8: STOP

Insertion at After given INFO Node

Step 1: Start

Step 2: IF AVAIL=NULL then PRINT OVERFLOW AND STOP

Step 3: Read DATA as information of node after which insertion will be made

Step 4: PTR=AVAIL

AVAIL= LINK(AVAIL)

READ INFO(PTR)

Step 5: CPT= FIRST

Step6: Repeat Step 7 while INFO(CPT)!=DATA

Step 7: CPT=Link(CPT)

Step 8: LINK(PTR)=LINK(CPT)

LINK(CPT) =PTR

Step 9: STOP

Insertion at before given INFO Node

Step 1: Start

Step 2: IF AVAIL=NULL then PRINT OVERFLOW AND STOP

Step 3: Read DATA as information of node before which insertion will be made

Step 4: PTR=AVAIL

AVAIL= LINK(AVAIL)

READ INFO(PTR) // New node of info to be inserted

Step 5: CPT= FIRST

TPT=FIRST

Step6: Repeat Step 7 while LINK(CPT)!=NULL

Step 7: if(INFO(CPT)==DATA) BREAK;

Otherwise, TPT=CPT;

CPT=LINK(CPT)

END While

Step 8: LINK(PTR)=LINK(TPT)

LINK(TPT) =PTR

Step 9: STOP

Insertion in the sorted List

Step 1: Start

Step 2: IF AVAIL=NULL then PRINT OVERFLOW AND STOP

Step 3: PTR=AVAIL

AVAIL= LINK(AVAIL)

READ INFO(PTR) // New node of info to be inserted

Step 4: CPT= FIRST

Step 5: Repeat Step 6 while INFO(CPT)<INFO(PTR)

Step 6: TPT=CPT

CPT=LINK(CPT)

Step 7: LINK(TPT)=PTR

LINK(PTR) =CPT

Step 9: STOP

Deletion in a Singly linked list

Underflow condition: if FIRST==NULL, write underflow and stop

After deletion: the deleted node will be inserted back to availability list at AVAIL position.

Let PTR points to the deleted node then the following step is used to insert in the availability list:

LINK(PTR)=AVAIL

AVAIL= PTR

Deletion is possible from following position:

i) From the beginning

ii) From the end

iii) If information of node is given

Deletion From the beginning

Step 1: Start

Step 2: If FIRST==NULL , Write underflow and STOP

Step 3: PTR=FIRST

FIRST=LINK(PTR)

Step 4: LINK(PTR)=AVAIL

AVAIL=PTR

Step 5: STOP

Deletion From the end

Step 1: Start

Step 2: If FIRST==NULL , Write underflow and STOP

Step 3: PTR=FIRST

Step 4: Repeat step 5 until LINK(PTR)!=NULL

Step 5: CPT=PTR

PTR= LINK(PTR)

Step 6: LINK(CPT)=NULL

Step 7: LINK(PTR)= AVAIL

AVAIL=PTR

Step 8: STOP

Deletion of the given INFO node

Step 1: Start

Step 2: If FIRST==NULL , Write underflow and STOP

Step 3: Read DATA as information of node to be deleted

Step 4: PTR=FIRST

Step 5: Repeat step 6 until INFO(PTR)!=DATA

Step 6: CPT=PTR

PTR= LINK(PTR)

Step 7: LINK(CPT)=LINK(PTR)

Step 8: LINK(PTR)= AVAIL

AVAIL=PTR

Step 9: STOP

Sorting the linked list

Step 1: Start

Step 2: PTR=FIRST

Step 3: Repeat step 4 to 8 while LINK(PTR)!=NULL

Step 4: CPT=LINK(PTR)

Step 5: Repeat Step 6 and 7 while CPT!=NULL

Step 6 : IF INFO(PTR)>INFO(CPT) then

TEMP=INFO(PTR)

INFO(PTR)=INFO(CPT)

INFO(CPT)=TEMP

STEP 7: CPT=LINK(CPT)

Step 8: PTR=LINK(PTR)

STEP 9: STOP

Searching a Node in Linked List

Step 1: Start

Step 2: Read DATA as information of node to be searched

Step 3: PTR=FIRST

Step 4: Repeat step 5 to 6 while INFO(PTR)!=DATA

Step 5: If INFO(PTR)==DATA then print “Search success” and GOTO Step 8

Step 6: PTR=LINK(PTR)

Step 7 : Print “ this info is not found”

STEP 8: STOP

Linked Queues

Insert operation: The insert operation append the queue by adding an element to the end of the queue. The new element will be the last element of the queue.

Step 1: Start

Step 2: Allocate the space for the new node PTR i.e,
Ptr = (struct node *) malloc (sizeof(struct node));

Step 3: SET INFO(PTR) = DATA

Step 4: IF FRONT == NULL
SET FRONT = REAR = PTR
SET LINK(FRONT)= LINK(REAR)= NULL
ELSE
SET LINK(REAR)= PTR
SET REAR = PTR
SET LINK(REAR)= NULL
[END OF IF]

Step 5: STOP

Contd..

Deletion operation: Deletion operation removes the element that is first inserted among all the queue elements. Firstly, we need to check either the list is empty or not. The condition front == NULL becomes true if the list is empty, in this case , we simply write underflow on the console and make exit.

Otherwise, we will delete the element that is pointed by the pointer front. For this purpose, copy the node pointed by the front pointer into the pointer PTR. Now, shift the front pointer, point to its next node and free the node pointed by the node PTR.

Step 1: START

Step 2: IF FRONT == NULL Write " Underflow “ Go to Step 5 [END OF IF]

Step 3: SET PTR = FRONT

Step 4: SET FRONT = LINK(FRONT)

Step 5: FREE PTR

Step 6: STOP

Linked STACK

PUSH operation: Inserting an Element to STACK

Step 1: Start

Step 2: Allocate the space for the new node PTR i.e,

Ptr = (struct node *) malloc (sizeof(struct node));

Step 3: IF TOP==NULL

SET LINK(PTR)=NULL

ELSE

SET LINK(PTR)=TOP

Step 4: TOP=PTR

Step 5: STOP

Contd..

POP operation: Deleting an Element from Stack.

Step 1: Start

Step 2: IF TOP==NULL PRINT “Stack is Empty” STOP

Step 3: Repeat Step 4 and 5 until LINK(TEMP) !=NULL

Step 4: TEMP=TOP

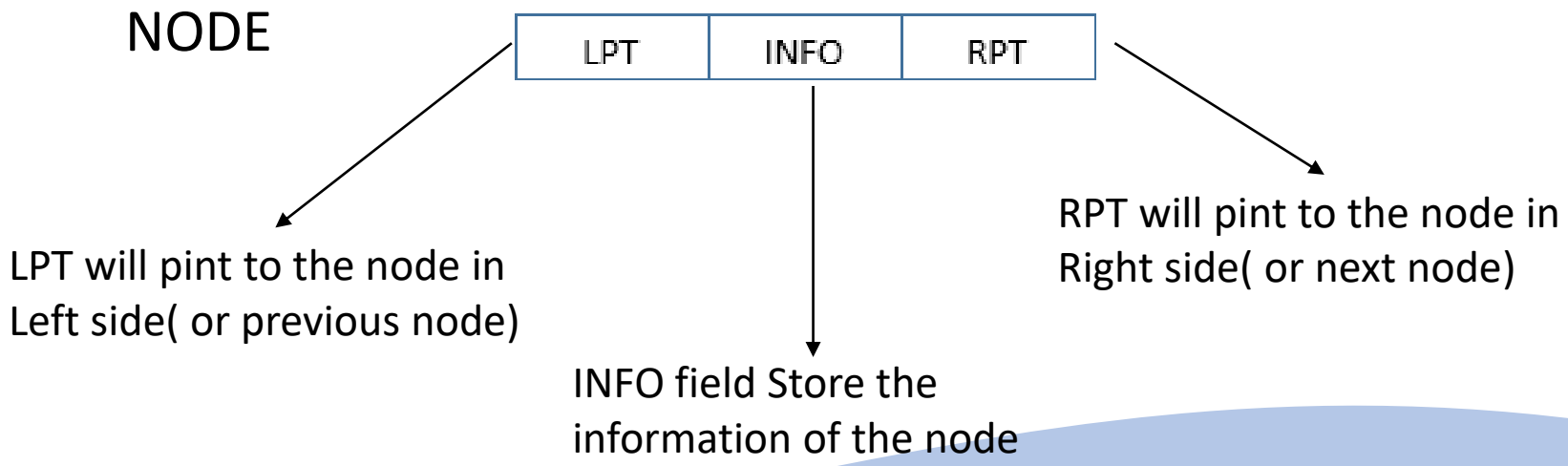
Step 5: PRINT INFO(TEMP)

SET TOP= LINK(TEMP)

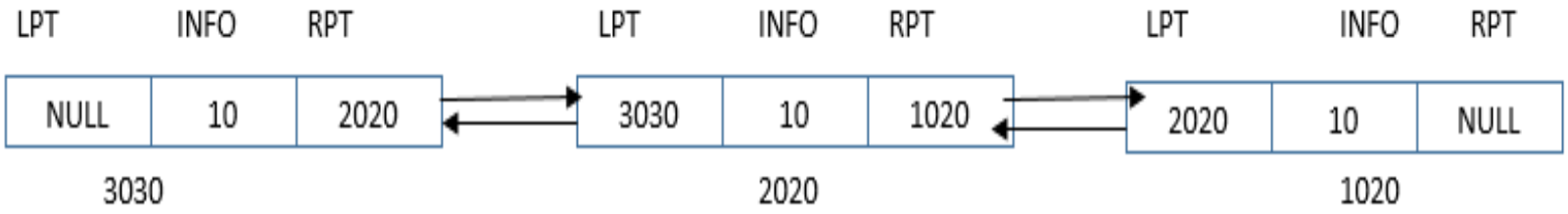
Step 6: STOP

Doubly Linked List

- Is also called two way Linked list and uses double set of pointers, one pointing to the next node and the other pointing to the preceding node.
- DLL is one in which all nodes are linked together by multiple links which help in accessing both the successor and the predecessor node for any arbitrary node within the list



A Doubly Linked List can be shown as follows:



Operation of DLL: Creation of DLL

Structure defined for doubly Linked List:

```
struct node
{       int info;
        struct node *rpt;
        struct node *lpt;
};
```

Step 1: Start

Step 2: PTR= AVAIL
AVAIL=RPT(AVAIL)
READ INFO(PTR)
LPT(PTR)=NULL
FIRST= PTR

Step 3: Repeat Step 4 While user choice 'Y'

Step 4: a)CPT= AVAIL
AVAIL=RPT(AVAIL)
READ INFO(PTR)

b) RPT(PTR)=CPT
LPT(CPT)=PTR
PTR= CPT

c) Input choice <Y/N> for more node

Step 5: RPT(PTR)=NULL

Step 6: STOP

Traversing of DLL

Forward Traversing:

Step 1: Start

Step 2: PTR=First

Step 3: Repeat step 4 and 5 while RPT(PTR)!=NULL

Step 4: PRINT INFO(PTR)

Step 5: PTR= RPT(PTR)

Step 6: Stop

Backward Traversing:

Step 1: Start

Step 2: PTR=First

Step 3: Repeat step 4 while RPT(PTR)!=NULL

Step 4: PTR=RPT(PTR)

Step 5: Repeat step 6 and 7 while LPT(PTR)!=NULL

Step 6: PRINT INFO(PTR)

Step 7: PTR= LPT(PTR)

Step 8: Stop

Insertion of Doubly Linked List

Insertion of Doubly Linked list can be done at:

a) At Beginning

b) At End

c) After a given Node

d) At in the sorted DLL

At the beginning:

Step 1: Start

Step 2: If AVAIL==NULL then Write "Overflow" and Stop.

Step 3: PTR=AVAIL

AVAIL=RPT(AVAIL)

Read INFO(PTR)

STEP 4: RPT(PTR)=FIRST

LPT(FIRST)=PTR

LPT(PTR)=NULL

FIRST=PTR

STEP 5: STOP

At the End:

Step 1: Start

Step 2: If AVAIL==NULL then Write "Overflow" and Stop.

Step 3: PTR=AVAIL

AVAIL=RPT(AVAIL)

Read INFO(PTR)

Step 4: CPT=FIRST

Step 5: Repeat Step 6 while RPT(CPT)!= NULL

STEP 6: CPT=RPT(CPT)

Step 7: RPT(CPT)=PTR

LPT(PTR)=CPT

RPT(PTR)=NULL

STEP 8: STOP

After given Node Information:

Step 1: Start

Step 2: If AVAIL==NULL then Write "Overflow" and Stop.

Step 3: PTR=AVAIL

AVAIL=RPT(AVAIL)

Read INFO(PTR)

Step 4: Read DATA as information after which insertion will be made

Step 5: CPT=FIRST

Step 6: Repeat Step 7 while INFO(CPT)!= DATA

Step 7: CPT= RPT(CPT)

STEP 8: TPT=RPT(CPT)

RPT(CPT)=PTR

LPT(PTR)=CPT

RPT(PTR)=TPT

LPT(TPT)=PTR

STEP 9: STOP

Insertion in the Sorted Doubly Linked List

Step 1: Start

Step 2: If AVAIL==NULL then Write "Overflow" and Stop.

Step 3: PTR=AVAIL

AVAIL=RPT(AVAIL)

Read INFO(PTR)

Step 4: Read DATA as information after which insertion will be made

Step 5: CPT=FIRST

Step 6: Repeat Step 7 while INFO(CPT)!= DATA

Step 7: CPT= RPT(CPT)

STEP 8: TPT=RPT(CPT)

RPT(CPT)=PTR

LPT(PTR)=CPT

RPT(PTR)=TPT

LPT(TPT)=PTR

STEP 9: STOP

Deletion of Doubly Linked List

Deletion of Doubly Linked list can be done :

a) From Beginning

b) From End

c) If a specific Node is given

Deletion from Beginning

Step 1: Start

Step 2: If FIRST == NULL, then write “UNDERFLOW” and STOP

Step 3: PTR=FIRST

First= RPT(PTR)

LPT(FIRST)= NULL

Step 4: RPT(PTR)= AVAIL

AVAIL= PTR

STEP 5: STOP

Deletion from End

Step 1: Start

Step 2: If FIRST == NULL, then write “UNDERFLOW” and STOP

Step 3: PTR=FIRST

Step 4: Repeat Step 5 while RPT(PTR) !=NULL

Step 5: PTR= RPT(PTR)

STEP 6: CPT=LPT(PTR)

RPT(CPT)=NULL

STEP 7: RPT(PTR)=AVAIL

AVAIL=PTR

STEP 8: STOP

Deletion FROM IF A SPECIFIC NODE IS GIVEN

Step 1: Start

Step 2: If FIRST == NULL, then write “UNDERFLOW” and STOP

STEP 3: Read DATA as information of node to be deleted

Step 4: PTR=FIRST

Step 5: Repeat Step 6 while INFO(PTR) !=DATA

Step 6: PTR= RPT(PTR)

STEP 7: CPT=LPT(PTR)

TPT=RPT(PTR)

RPT(CPT)=TPT

LPT(TPT)=CPT

STEP 8: RPT(PTR)=AVAIL

AVAIL=PTR

STEP 9: STOP

Any Query?