LINKED LISTS: Insertion Algorithms

Prepare your NICE and INTERACTIVE presentation slides on the following topics individually:

- **Insertion Algorithms in Linked List**
- **Inserting element at the Beginning of a Linked List**
- **Inserting element after a Given node of a Linked List**
- **Inserting element into a sorted Linked List.**

Presentation Schedule: Next Class Period
Presentation Duration: 7-8 minutes + 2-3 minutes QA.

Good Luck …!!

© Dr. Md. Golam Rashed, Assoc. Professor, Dept. of ICE, RU

LINKED LISTS: Insertion Algorithms

- ✓ Algorithms which inserts nodes into linked lists come up in various situations :
  - ➢ Inserts a node at the beginning of the list,
  - ➢ Inserts a node after the node with a given location, and
  - ➢ Inserts a node into a sorted list.
- ✓ All the algorithms:
  - ➢ assume that linked list is in memory in the form

    LIST(INFO, LINK, START, AVAIL)
  - ➢ Have a variable **ITEM** which contains the new information to be added to the list.

© Dr. Md. Golam Rashed, Assoc. Professor, Dept. of ICE, RU

LINKED LISTS: Insertion Algorithms

Since all the insertion algorithm will use a node in the AVAIL list, all of the algorithm will include the following steps:

- Checking to see if space is available in the AVAIL list. If not, that is,
- If **AVAIL=NULL**, then the algorithm will print the message

  OVERFLOW.

- Removing the first node from the AVAIL list. Using the variable NEW to keep track of the location of the new node.

  NEW:=AVAIL, AVAIL:=LINK[AVAIL]

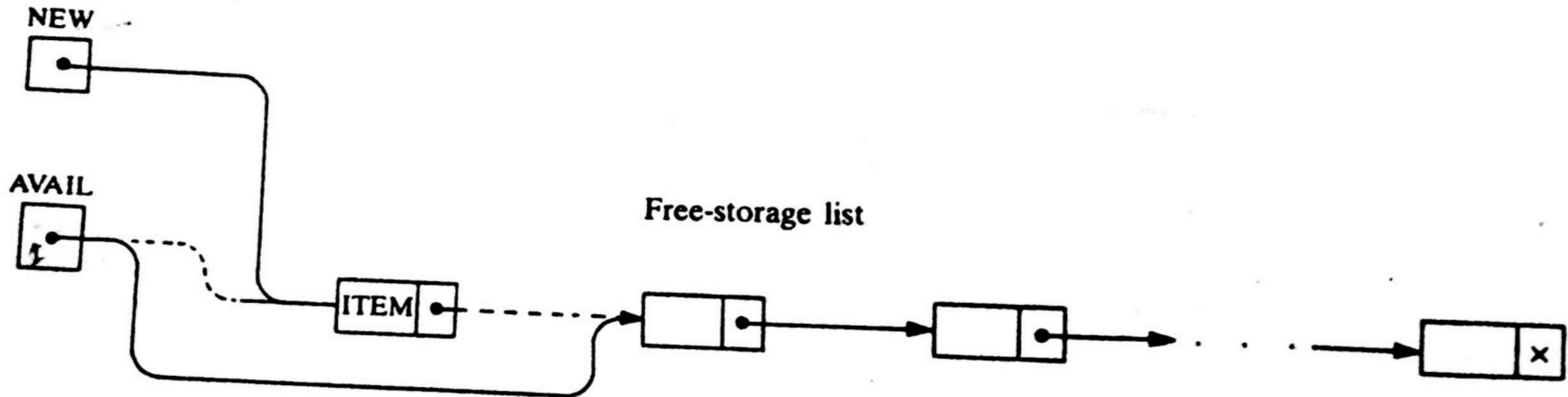
- Copying new information into the new node. i.e.,

  INFO[NEW]=ITEM

© Dr. Md. Golam Rashed, Assoc. Professor, Dept. of ICE, RU

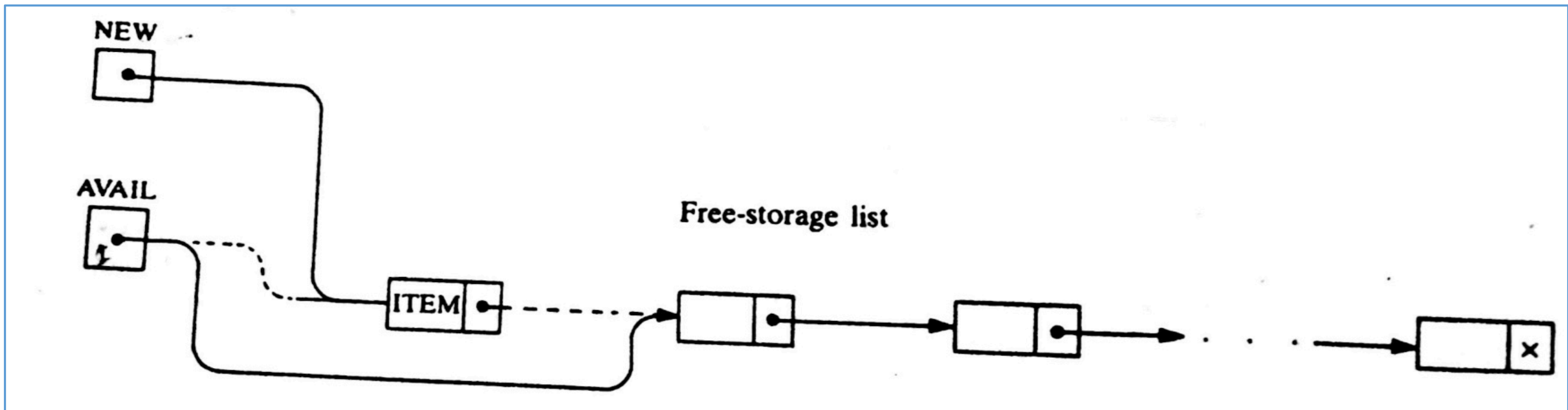
LINKED LISTS: Insertion Algorithms

ICE 2261



LINKED LISTS: Inserting at the Beginning of a list

ICE 2261



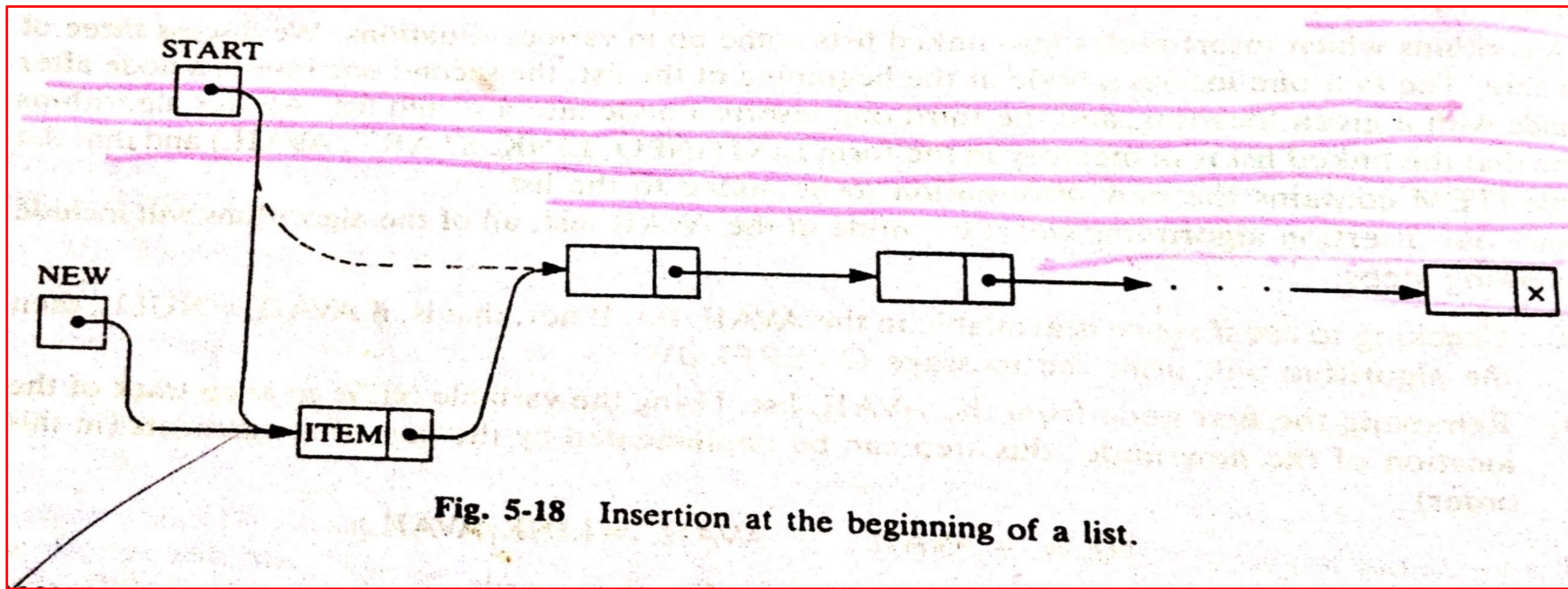


Fig. 5-18 Insertion at the beginning of a list.

© Dr. Md. Golam Rashed, Assoc. Professor, Dept. of ICE, RU

LINKED LISTS: Inserting at the Beginning ….ALG

INSFIRST (INFO, LINK, START, AVAIL,ITEM)

1. [OVERFLOW?] If AVAIL=NULL, then Write: OVERFLOW, and Exit.
2. [Remove first node from AVAIL list.]

   Set NEW:=AVAIL and AVAIL:=LINK[AVAIL]
3. Set INFO[NEW]:=ITEM.[Copies new data into new node.]
4. Set LINK[NEW]:=START.[New Node now points to original first node.]
5. Set START:=NEW. [Change START so it points to the new node.]
6. Exit.

LINKED LISTS: Inserting at the Beginning ….Exmp

ALG: 11
GEOM: 5
AVAIL: 9

	TEST	LINK
1		16
2	74	14
3		1
4	82	0
5	84	12
6	78	0
7	74	8
8	100	13
9		10
10		3
11	88	2
12	62	7
13	74	6
14	93	4
15		0
16		15

ALG: 11
GEOM: 9
AVAIL: 10

	TEST	LINK
1		16
2	74	14
3		1
4	82	0
5	84	12
6	78	0
7	74	8
8	100	13
9	75	5
10		3
11	88	2
12	62	7
13	74	6
14	93	4
15		0
16		15

© Dr. Md. Golam Rashed, Assoc. Professor, Dept. of ICE, RU

LINKED LISTS: Inserting after a given node

INSLOC (INFO, LINK, START, AVAIL, LOC, ITEM)

1. [OVERFLOW?] If AVAIL=NULL, then Write: OVERFLOW, and Exit.
2. [Remove first node from AVAIL list.]

   Set NEW:=AVAIL and AVAIL:=LINK[AVAIL]
3. Set INFO[NEW]:=ITEM.[Copies new data into new node.]
4. If LOC=NULL, then [Insert as first node.]

   Set LINK[NEW]:=START and START:=NEW.

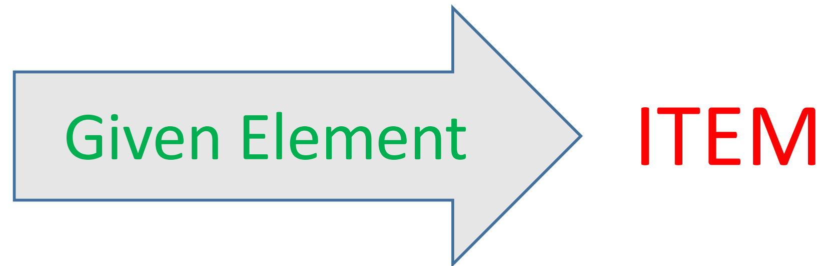
   Else: [Insert after node with location LOC.]

   Set LINK[NEW]:=LINK[LOC] and LINK[LOC]:=NEW.

   [END if If structure.]
5. Exit.

© Dr. Md. Golam Rashed, Assoc. Professor, Dept. of ICE, RU

LINKED LISTS: Inserting into a Sorted Linked List



ITEM must be inserted between nodes A and B so that

INFO[A] < ITEM <= INFO[B]

Thus,

- Firstly we should write an Algorithm to find the location, LOC, where is suitable to insert the **ITEM** in the sorted linked list.
- Secondly, we have to write the algorithm to insert to the found Location.

© Dr. Md. Golam Rashed, Assoc. Professor, Dept. of ICE, RU

LINKED LISTS: Find LOC in a Sorted Linked List

ICE 2261

FINDA(INFO, LINK, START, ITEM, LOC)

1. [List empty?] If START=NULL, then Set LOC:=NULL, and Return.
2. [Special Case?] If ITEM<INFO[START], then: Set LOC:=NULL, and, Return.
3. Set SAVE:=START and PTR:=LINK[START].[Initialize pointers]
4. Repeat Steps 5 and 6 while PTR $\neq$ NULL.
5. If ITEM<INFO[PTR], then:

   Set LOC:=SAVE, and Return.

   [End of If structure]

6. Set SAVE:=PTR and PTR:=LINK[PTR]. [Update pointers.]

   [End of Step 4 loop.]

7. Set LOC:= SAVE.
8. Return

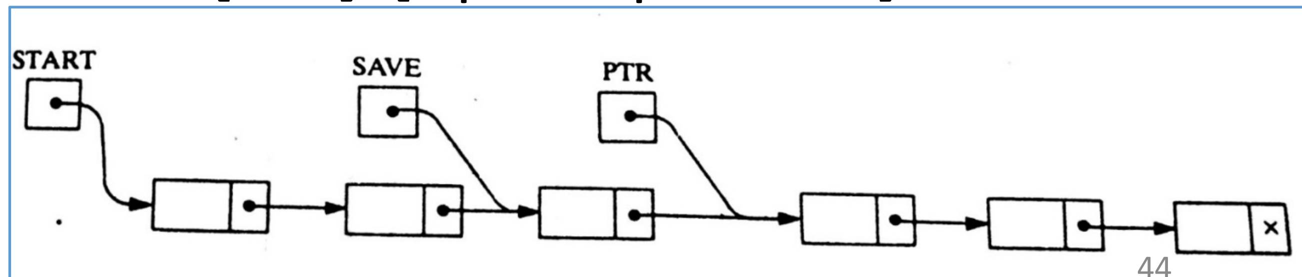


Fig. 5-20

© Dr. Md. Golam Rashed, Assoc. Professor, Dept. of ICE, RU

LINKED LISTS: Insert ITEM into a Sorted Linked List

INSSRT(INFO, LINK, START, AVAIL, ITEM)

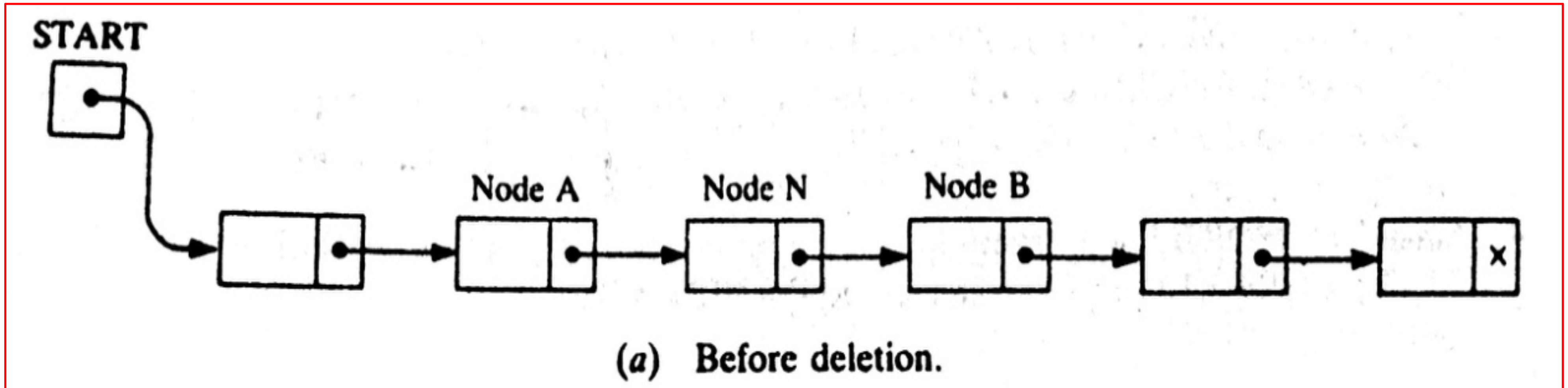
1. [Use **FINDA(INFO, LINK, START, ITEM, LOC)** to find the location of the proceeding ITEM]
2. Use **INSLOC (INFO, LINK, START, AVAIL, LOC, ITEM)** to insert ITEM after the node with location LOC.]
3. Exit.

© Dr. Md. Golam Rashed, Assoc. Professor, Dept. of ICE, RU

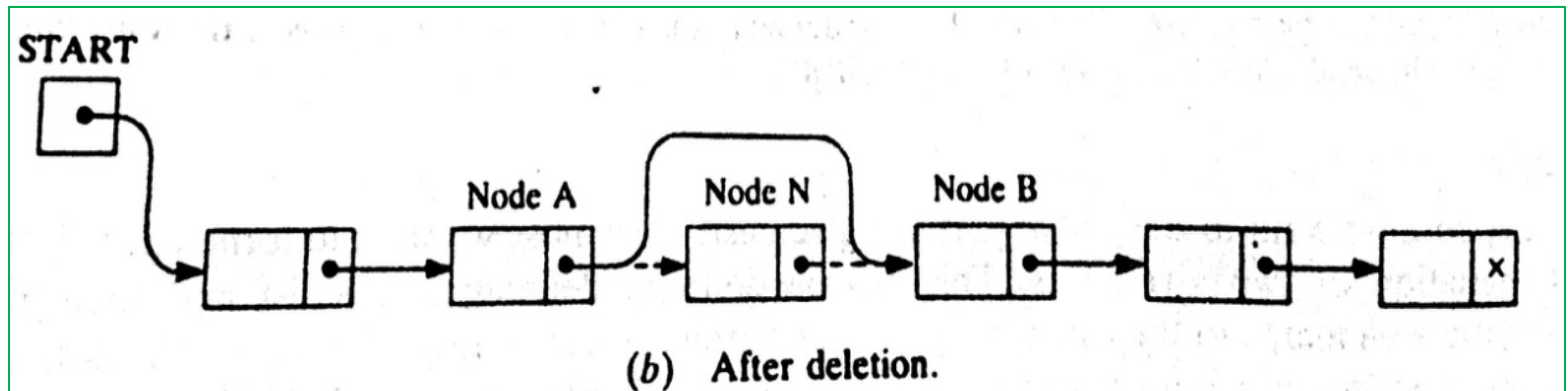
LINKED LISTS: Deletion from a Linked List

ICE 2261

Let LIST be a linked list with a node N between nodes A and B.



(a) Before deletion.



(b) After deletion.

Here we don't care about the future of the deleted nodes of the linked list

© Dr. Md. Golam Rashed, Assoc. Professor, Dept. of ICE, RU

LINKED LISTS: Deletion from a Linked List

ICE 2261

More Exact procedure:

START

Data list

Node A

Node N

Node B

AVAIL

Free-storage list

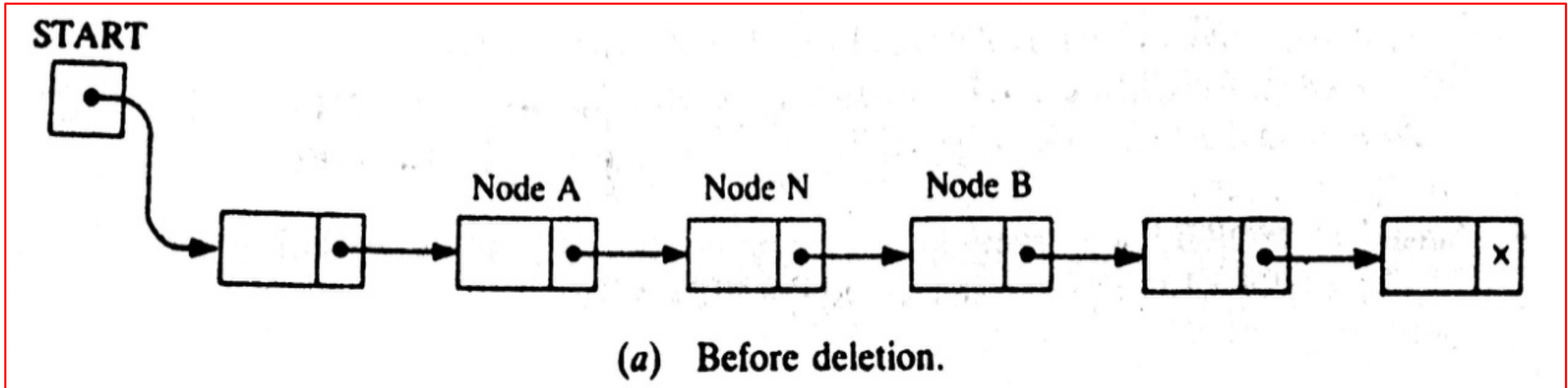
Three pointer fields are changed:

- The nextpointer field of node A now points to node B.
- The nextpointer field of N now points to the original first node in the free pool.
- AVAIL now points to the deleted node N.

LINKED LISTS: Deletion from a Linked List

ICE 2261

There are also TWO special cases:



(a) Before deletion.

- If the deleted node N is the first node in the list
  - ✓ **START will point to the node B**
- If the deleted node N is the last node in the list
  - ✓ **Node A will contain the NULL pointer**

© Dr. Md. Golam Rashed, Assoc. Professor, Dept. of ICE, RU

LINKED LISTS: Deletion Algorithms

Algorithms which delete nodes from linked lists come up in various situations:

- The first one deletes the node following a given node, and
- The second one deletes the node with a given ITEM of information

All algorithms assume that linked list is in memory in the form

LIST(INFO, LINK, START, AVAIL)

All the algorithms will return the memory space of the deleted node N to the beginning of the AVAIL list.

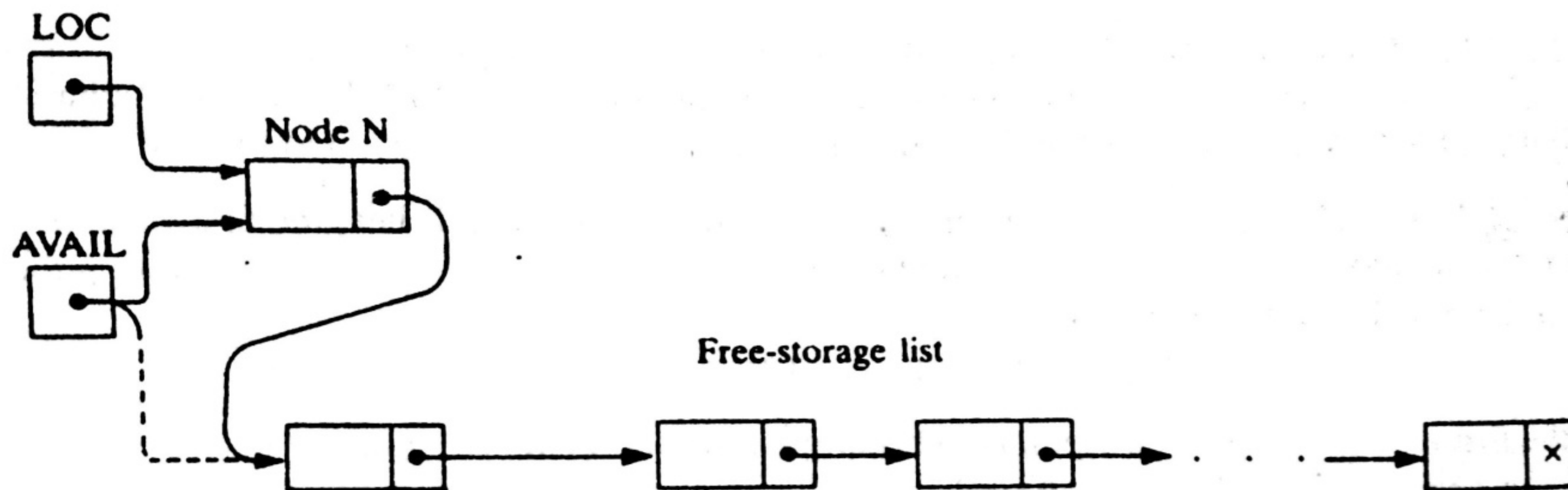


Fig. 5-25 LINK[LOC] := AVAIL and AVAIL := LOC.

LINKED LISTS:

ICE 2261

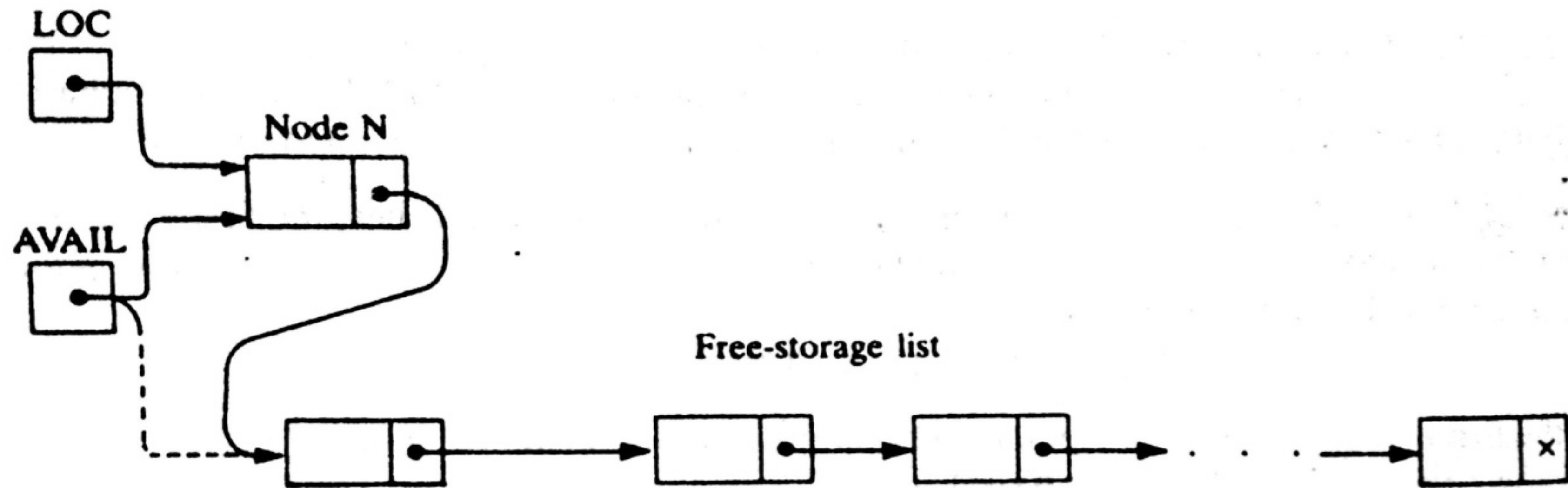


Fig. 5-25 LINK[LOC] := AVAIL and AVAIL := LOC.

All algorithms will include the following pair of assignments where LOC is the location of the deleted node N:

LINK[LOC]:=AVAIL and then AVAIL:=LOC

© Dr. Md. Golam Rashed, Assoc. Professor, Dept. of ICE, RU

LINKED LISTS: Deleting the node following a Given Node

DEL(INFO, LINK, START, AVAIL, LOC, LOCP)

(it deletes the node N with location LOC, LOCP is the location of the node which precedes N)

Step 1. If LOCP=NULL, then:

Set START:=LINK[START]. [Delete first node.]

Else:

Set LINK[LOCP]:=LINK[LOC]. [Deletes node N.]

[End of If structure]

Step 2. [Return deleted node to the AVAIL list.]

Set LINK[LOC]:=AVAIL and AVAIL:=LOC.

Step 3. Exit.

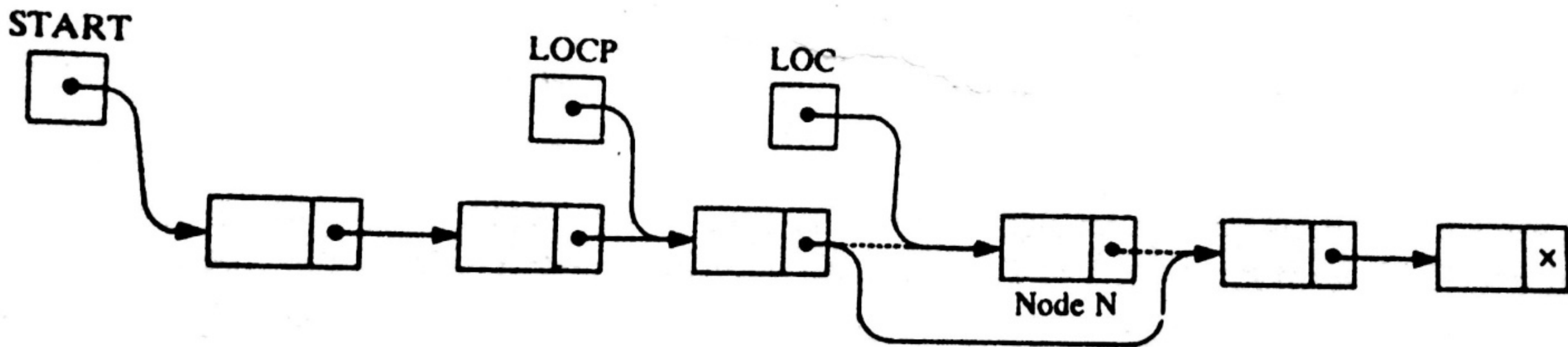


Fig. 5-27. LINK[LOCP]:= LINK[LOC].

LINKED LISTS: Deleting the node with a given ITEM

- Let LIST be a linked list in memory.
- Suppose we are given an ITEM of Information and we want to delete from the LIST the first node which contain ITEM.
- The algorithm has TWO **PARTS**

  - First we give a procedure which finds the location LOC of the node N containing ITEM, and the location LOCP of the node preceding node N.
    - If N is the first node with ITEM, we set LOCP=NULL
    - If ITEM does not appear in LIST, we set LOC=NULL

  - Traverse the list, using pointer variable PTR and comparing ITEM with INFO[PTR]at each node.
    - While track the location of the preceding node by using a pointer variable SAVE. Thus

SAVE:= PTR and PTR:= LINK[PTR]

© Dr. Md. Golam Rashed, Assoc. Professor, Dept. of ICE, RU

FINDB (INFO, LINK, START, ITEM, LOC, LOCP)

Step 1. [List Empty?] If Start=NULL, then:
Set LOC:=NULL and LOCP:=NULL, and Return.
[End of If Structure]

Step 2. [ITEM in first node?] If INFO[START]=ITEM, then:
Set LOC:=START and LOCP=NULL, and Return.
[End of If structure]

Step 3. Set SAVE:=START and PTR:=LINK[START]. [Initialize pointers]

Step 4. Repeat Steps 5 and 6 while PTR$\neq$ NULL.

Step 5. If INFO[PTR]=ITEM, then:
Set LOC:=PTR and LOCP:=SAVE, and Return.
[End of If structure]

Step 6. Set SAVE:=PTR and PTR:=LINK[PTR]. [Update pointers]
[End of Step 4 loop]

STEP 7. Set LOC:=NULL. [Search Unsuccessful.]

Step 8. Return

© Dr. Md. Golam Rashed, Assoc. Professor, Dept. of ICE, RU

DELETE (INFO, LINK, START, AVAIL, ITEM)

Step 1. Call **FINDB(INFO, LINK, START, ITEM, LOC, LOCP)**

Step 2. If LOC=NULL, then: Write: ITEM not in list, and Exit.

Step 3. [Delete node.]

If LOCP=NULL, then

Set Start:=LINK[START]. [Delete fist node]

Else:

Set LINK[LOCP]:=LINK[LOC].

[End of If structure]

Step 4. [Return deleted node to the AVAIL list.]

Set LINK[LOC]:=AVAIL and AVAIL:=LOC.

Step 5. Exit

© Dr. Md. Golam Rashed, Assoc. Professor, Dept. of ICE, RU

LINKED LISTS:

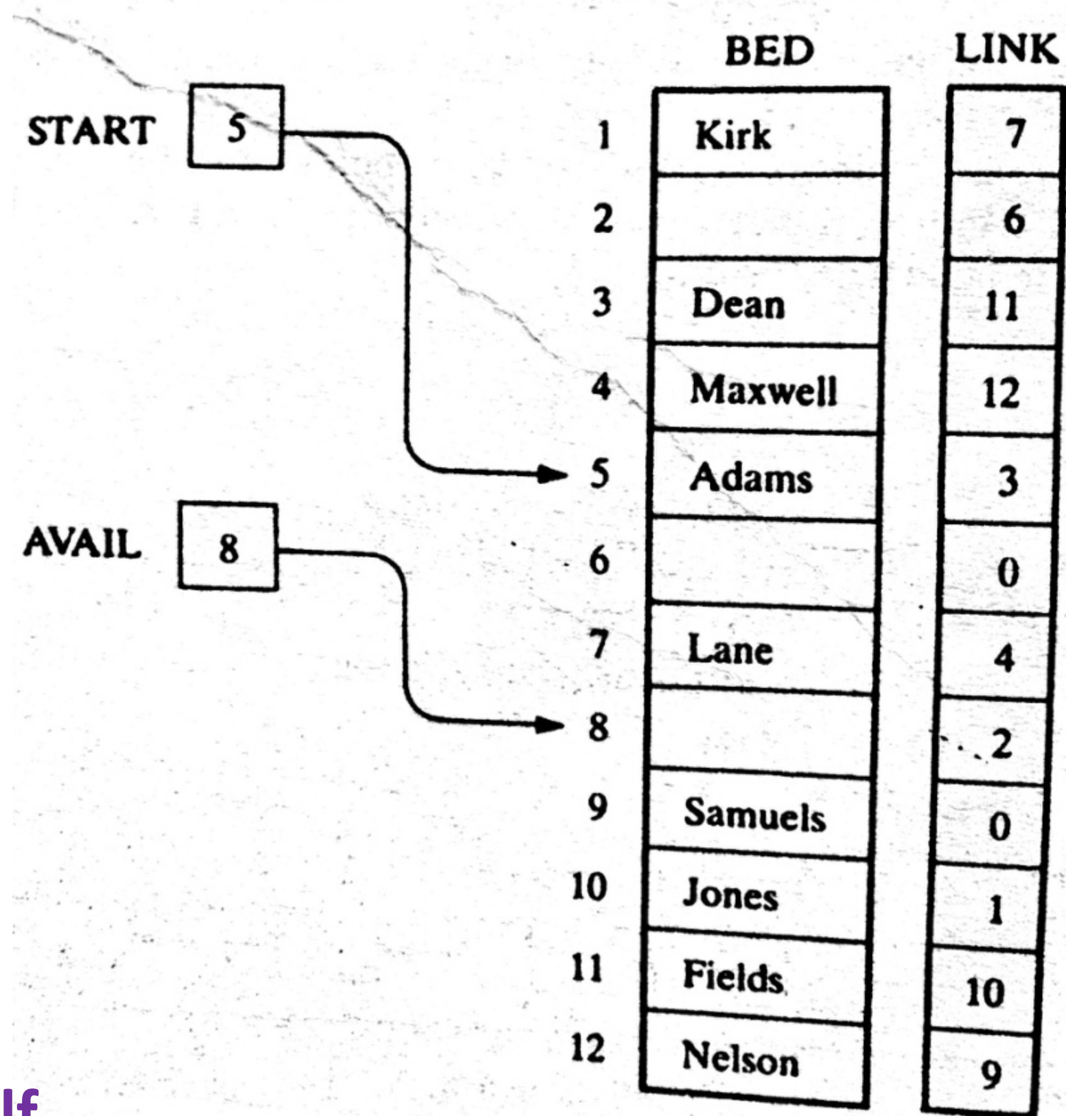
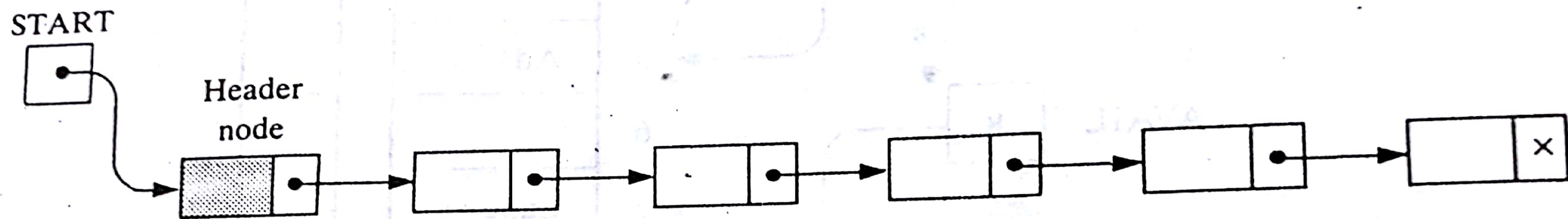


Fig. 5-28

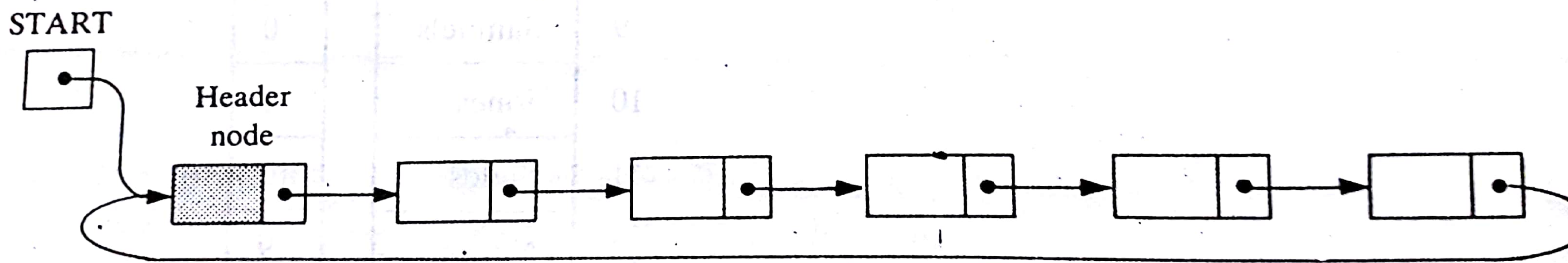
Example 5.17 Teach Yourself

Header Linked LISTS

- A header lined list is a linked list which always contains a special node, called the ***header node***, at the beginning of the list.
- The following are two kinds of widely used header lists:
  - **A Grounded Header List**:
  - **A Circular List:**



(*a*) Grounded header list.

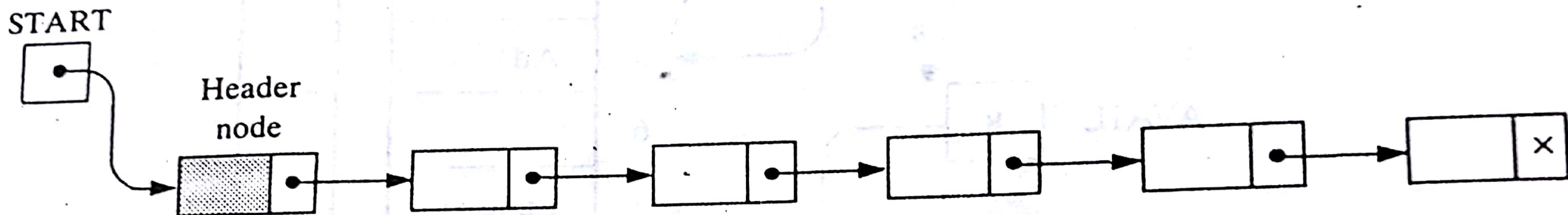


(*b*) Circular header list.

Header Linked LISTS

Grounded Header List: is a header list where the last node contains the null pointer.

- Observe that, the list pointer START always points to the header node.
- Accordingly, LINK[START] =NULL indicates that a grounded header list is empty.



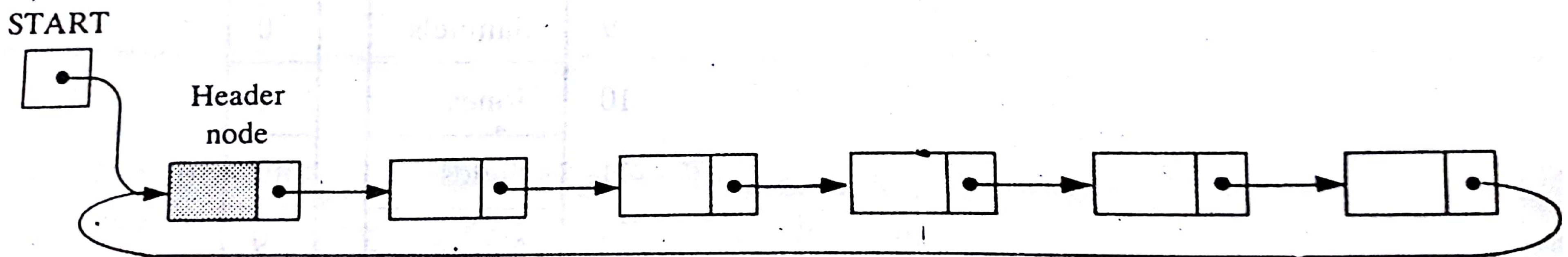
(*a*) Grounded header list.

© Dr. Md. Golam Rashed, Assoc. Professor, Dept. of ICE, RU

Header Linked LISTS

A Circular Header List: is a header list where the last node points back to the header node.

- Observe that, the list pointer START always points to the header node.
- LINK[START]=START indicates that a circular header list is empty.



© Dr. Md. Golam Rashed, Assoc. Professor, Dept. of ICE, RU

Header Linked LISTS: Traversing

Traversing a Circular Header List(INFO, LINK, START)

Step1. Set PTR:=LINK[START].[Initialize pointer PTR.]

Step2. Repeat Steps 3 and 4 while PTR ≠ START

Step3. Apply PROCESS to INFO[PTR]

Step4. Set PTR:=LINK[PTR]. [PTR now point to the next node.]

[End of step 2 loop]

Step5. Exit.

LINKED_LIST_Traversing(INFO, LINK, START)

Step1. Set PTR:=START.[Initialize pointer PTR.]

Step2. Repeat Steps 3 and 4 while PTR ≠ NULL

Step3. Apply PROCESS to INFO[PTR]

Step4. Set PTR:=LINK[PTR]. [PTR now point to the next node.]

[End of step 2 loop]

Step5. Exit.

© Dr. Md. Golam Rashed, Assoc. Professor, Dept. of ICE, RU

Header Linked LISTS:

(Suppose LIST is a header linked list in memory, and suppose a specific ITEM of information is given)

SRCHHL(INFO, LINK, START, ITEM, LOC)

[Finds the location of the first node in LIST which contains ITEM in a circular header list]

Step1. Set PTR:= LIST [START].

Step2. Repeat while INFO[PTR] ≠ITEM and PTR ≠ START:

Set PRT := LINK [PTR]. [PTR now points to the next node.]

[End of Loop]

Step3. If INFO[PTR]=ITEM, then:

Set LOC:= PTR.

ELSE:

SET LOC:=NULL.

[End of If structure]

Step4. Exit.

The two test which control the searching loop were not performed at the same time in the ordinary linked lists.

Linked LISTS: Searching

SEARCH (INFO, LINK, START, ITEM, LOC)

Step1. Set PTR:=START

Step2. Repeat Step 3 while PTR ≠ NULL

Step3. If ITEM=INFO [PTR], then:

Set LOC:=PTR, and Exit.

Else:

Set LOC:=LINK[PTR]. [PTR now points to the next node.]

[End of If structure.]

[End of Step 2 loop.]

Step4. [Search is unsuccessful.] Set LOC:=NULL.

Step5. Exit.

© Dr. Md. Golam Rashed, Assoc. Professor, Dept. of ICE, RU

Circular Header Linked LISTS: Find Location

ICE 2261

FINDBHL (INFO, LINK, START, ITEM, LOC, LOCP)

Finds the location LOC of the first node N which contains ITEm and also the location LOCP of the node preceding N.)

Step1. Set SAVE:=START and PTR:= LINK[START] [Initialize pointers]

Step2. Repeat while INFO[PTR] ≠ITEM and PTR ≠ START:

Set SAVE := PTR and PTR:= LINK [PTR]. [Update pointers.]

[End of Loop]

Step3. If INFO[PTR]=ITEM, then:

Set LOC:= PTR and LOCP:=SAVE.

ELSE:

SET LOC:=NULL and LOCP:=SAVE

[End of If structure]

Step4. Exit.

© Dr. Md. Golam Rashed, Assoc. Professor, Dept. of ICE, RU

Circular Header Linked LISTS: Delete

DELLOCHL (INFO, LINK, START, AVAIL, ITEM)

(Deletes the first nodes N which contains ITEM when LIST is a Circular Header List.

Step1. Call FINDBHL (INFO, LINK. START, ITEM, LOC, LOCP)

Step2. If LOC=NULL, then: write: ITEM not in list, and Exit.

Step3. Set LOC[LOCP]:=LINK [LOC] **[Delete Node.]**

Step4. [Return deleted node to the AVAIL list.]

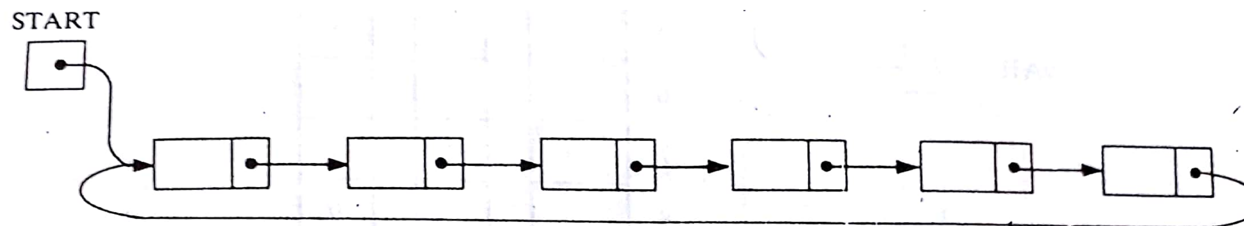
Set LINK[LOC]:=AVAIL and AVAIL:=LOC.

Step5. Exit

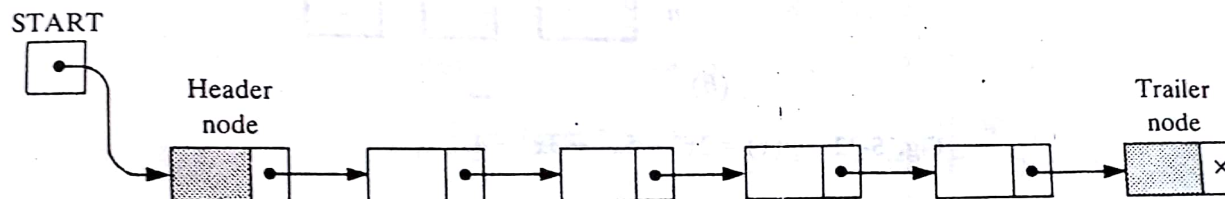
© Dr. Md. Golam Rashed, Assoc. Professor, Dept. of ICE, RU

TWO other variations of LINKED LIST

1. A linked list whose last node points back to the first node instead of containing the null pointer, called a *circular list*.
2. A linked list which contain both a special header node at the beginning of the list and a special trailer node at the end of the list.



(a) Circular linked list.



(b) Linked list with header and trailer nodes.

TWO-WAY LISTS:

A two-way list is a linear collection of data elements, called nodes, where each node N is divided into three parts:

(1) An information field INFO which contains the data of N

(2) A pointer field FORW which contains the location of the next node in the list

(3) A pointer field BACK which contain the location of the preceding node in the list.

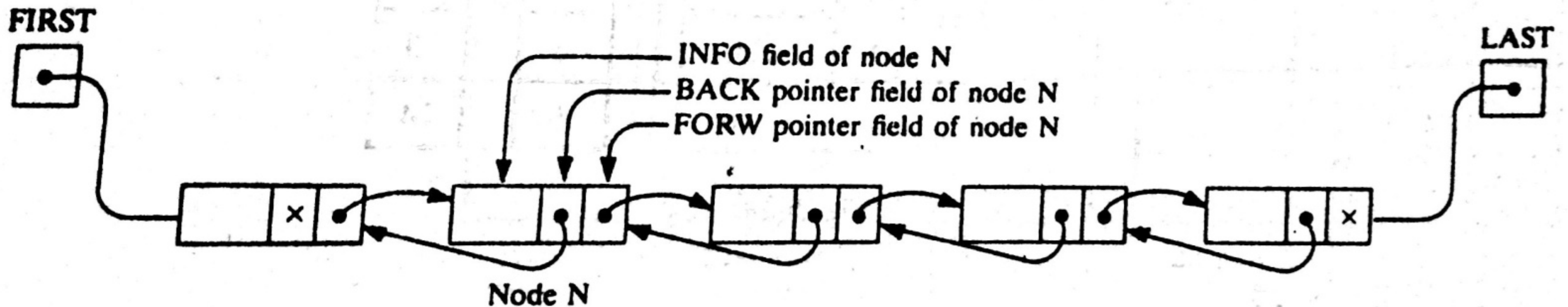


Fig. 5-33 Two-way list.

TWO-WAY LISTS: Operations

ICE 2261

Deleting:

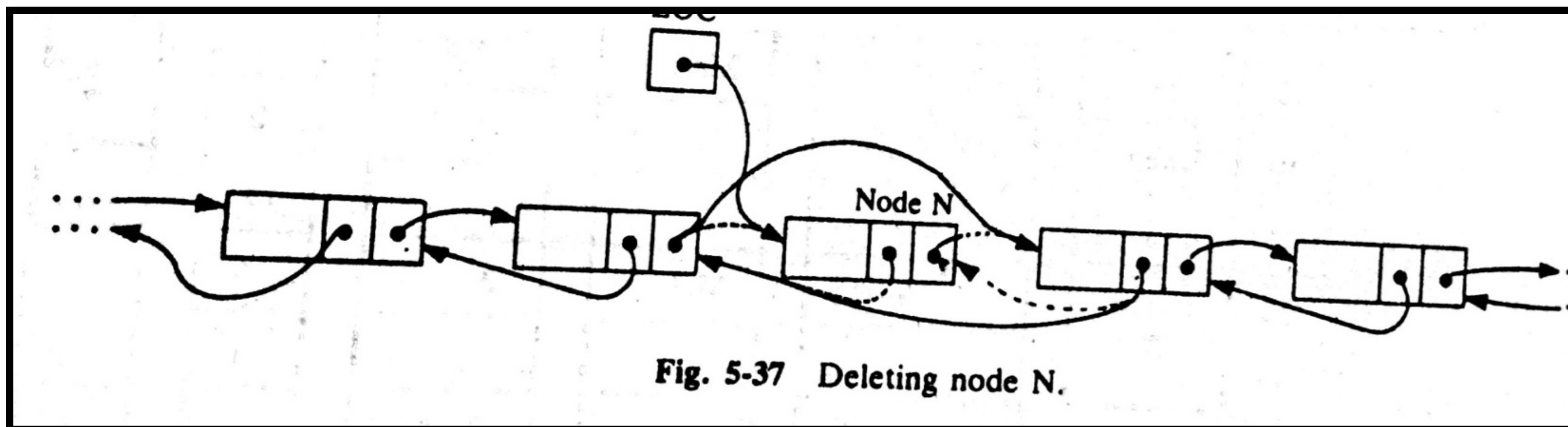


Fig. 5-37 Deleting node N.

Inserting:

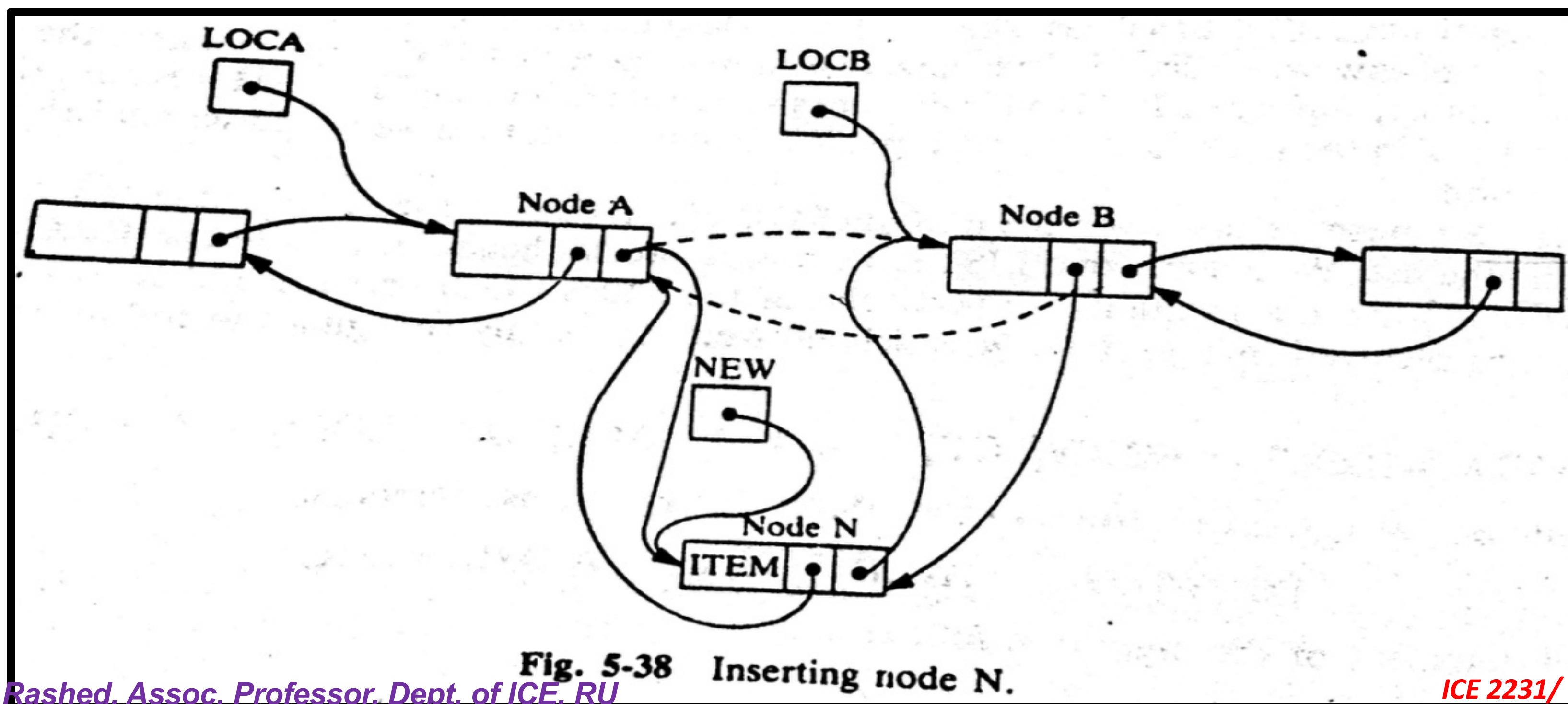


Fig. 5-38 Inserting node N.