Data Structure

تركيب بيانات
الفرقة الثالثة علوم إحصاء وعلوم الحاسب

By

Dr. Reda Elbarougy

د/ رضا الباروجى

Lecturer of computer sciences

In Mathematics Department

Faculty of Science

Damietta University

رقم المحاضرة

رقم المحاضرة	التاريخ	ملاحظات
المحاضرة 1	2020-02	مقدمة ـــ الفصل الاول
المحاضرة 2	2020-03-03	اساسيات تحليل الخورازميات
المحاضرة 3	2020-03-10	
المحاضرة 4	2020-03-17	
المحاضرة 5	2020-03-24	
المحاضرة 6	2020-03-31	
المحاضرة 7		
المحاضرة 8		
المحاضرة 9		
المحاضرة 10		
المحاضرة 11		

Chapter 5:
Linked Lists part II

أولا الإضافة

Inserting at the beginning of a list الاضافة فى البداية

Inserting after a Given Node الاضافة بعد عنصر معين

Inserting Into a Sorted linked list الاضافة فى قائمة مرتبة

Chapter 5: Linked Lists Part II

5.7. Insertion into a linked list

1. Example 5.13
2. Insertion Algorithms
    1. Inserting at the beginning of a list
    2. Algorithm 5.4 and example 5.14
    3. Inserting after a Given Node
    4. Algorithm 5.5
    5. Inserting Into a Sorted linked list
    6. Procedure 5.6, Algorithm 5.7, Ex. 5.15, Copying

5.7 Insertion into a Linked List

5.7 Insertion into a Linked List

- Let LIST be a linked list with successive nodes A and B as pictured in Fig. 5-14(a).
- Suppose a node N is to be inserted into the list between nodes A and B.

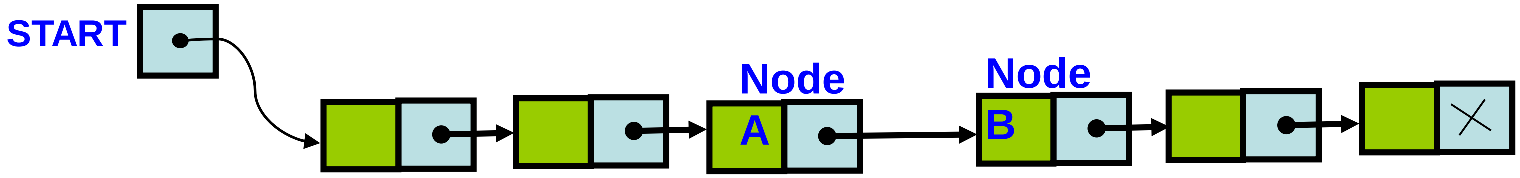


Fig. 5-14 (a) Before insertion

Insertion into a linked list

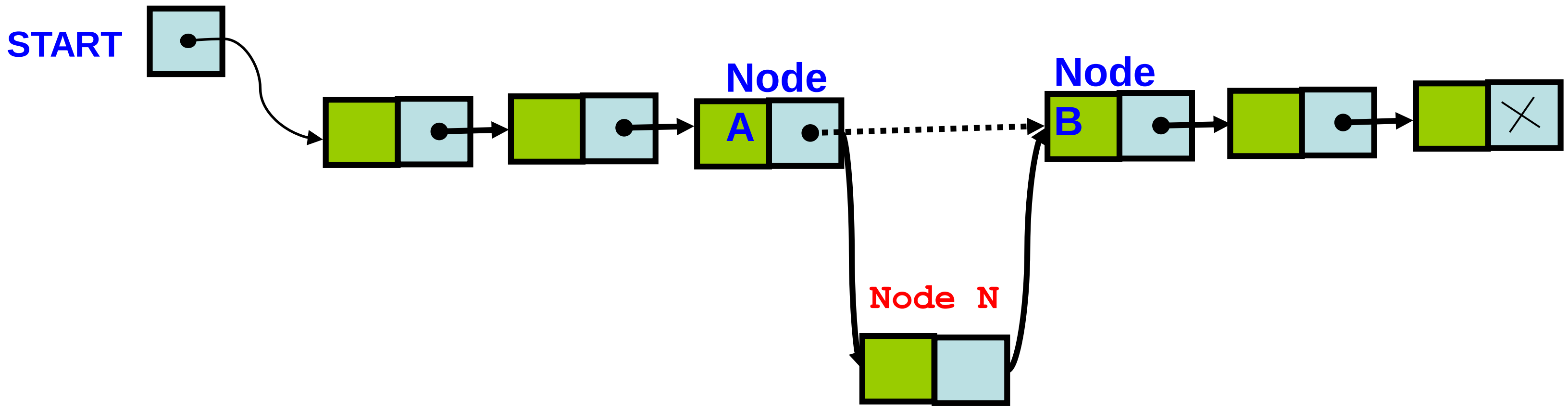


Fig. 5-14 (b) After insertion

- Node A now points to the new node N, and the node N points to node B, to which A previously pointed.

5.7 Insertion into a linked list

- Suppose our linked list is maintained in memory in the form
  LIST(INFO, LINK, START, AVAIL)
- Figure 5-14 does not take into account that the memory space for the new node N will come from the AVAIL list.
- Node N is to be inserted in to the list between nodes A and B

Three pointer fields are changed as follows:

1. The next pointer field of node A now points to the new node N, to which AVAIL previously pointed.
2. AVAIL now point to the second node in the free pool, to which node N previously pointed.
3. The next pointer field of node N now points to node B, to which node A previously pointed.

5.7 Insertion into a linked list

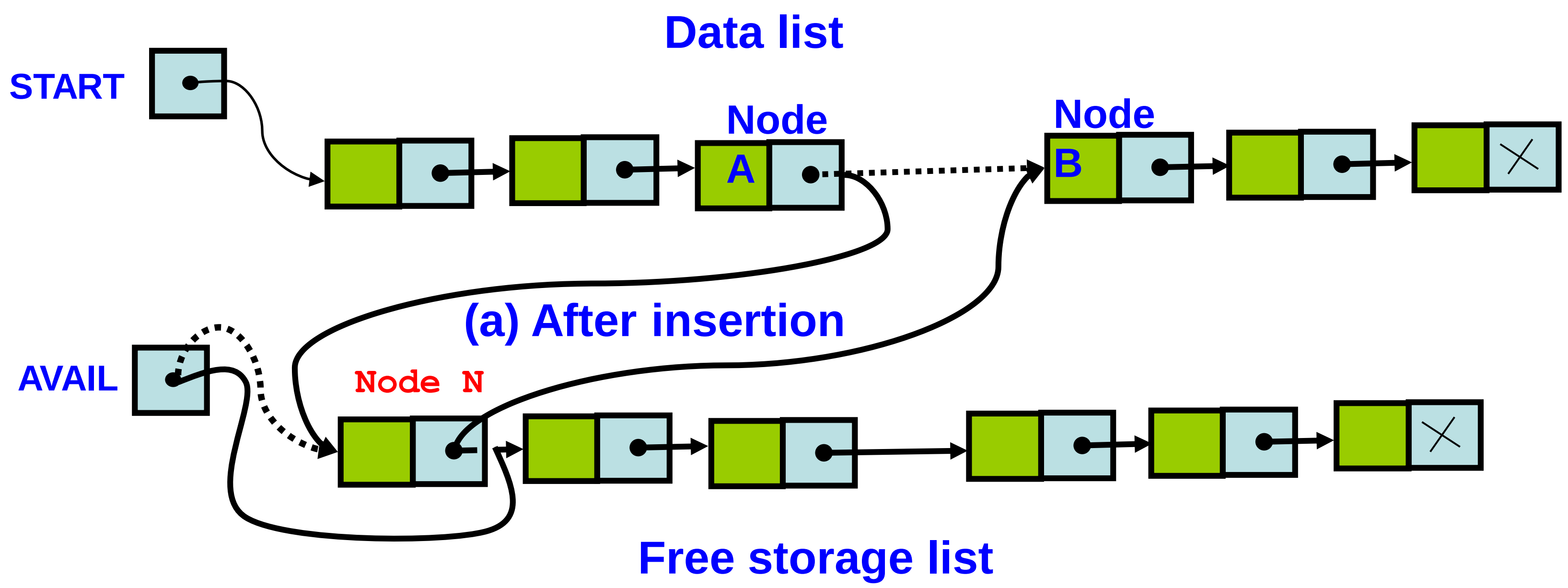


Fig. 5-15

EXAMPLE 5.13

(a) Consider Fig. 5-9, the alphabetical list of patients in a ward. Suppose a patient Hughes is admitted to the ward.

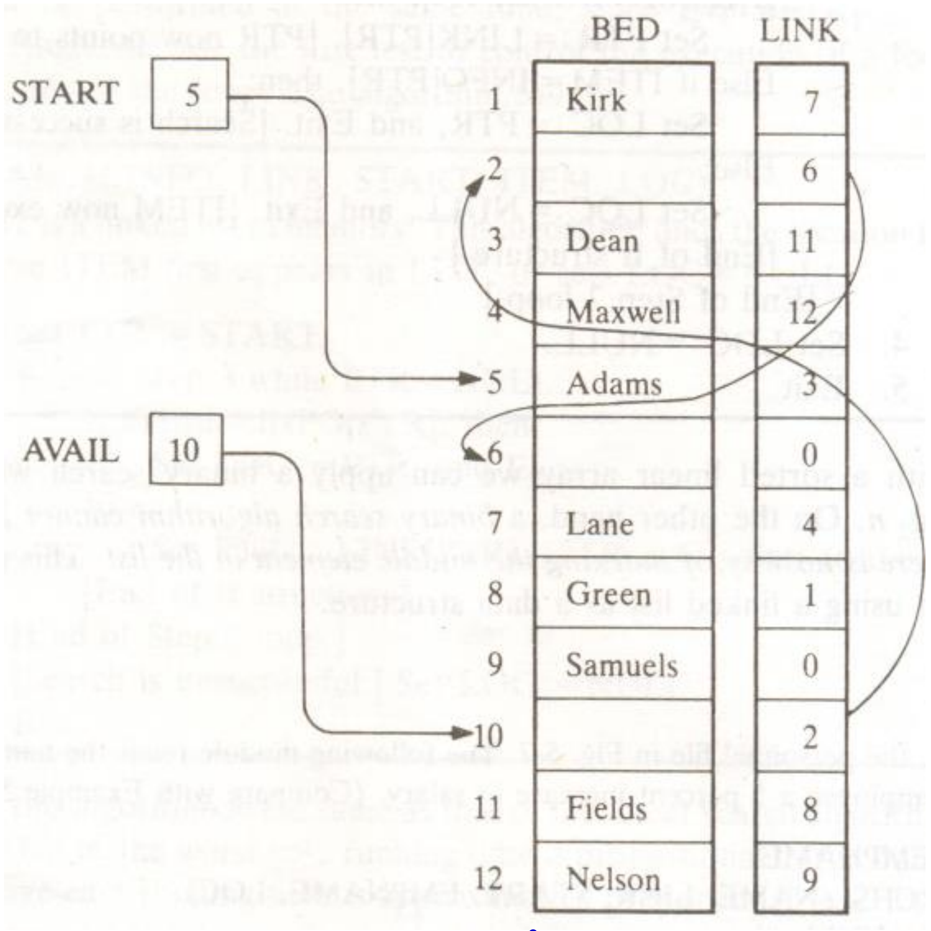


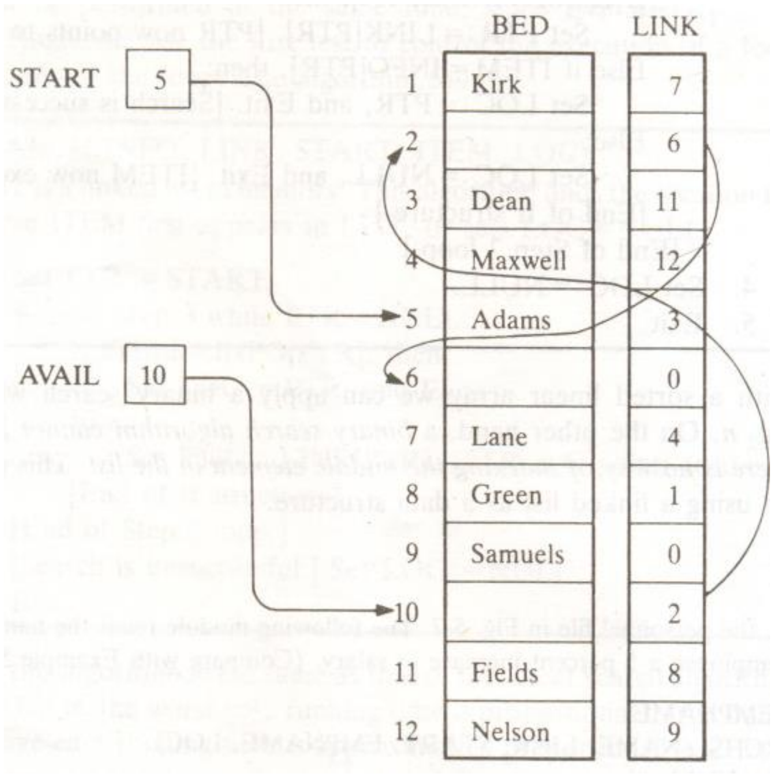
Fig. 5-9

Green

Hughes

Kirk

EXAMPLE 5.13

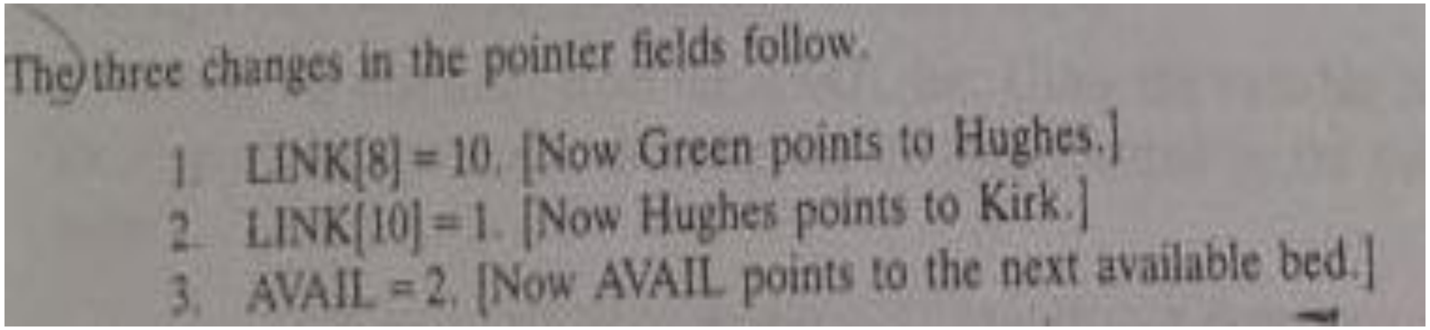


	BED	LINK
1	Kirk	7
2		6
3	Dean	11
4	Maxwell	12
5	Adams	3
6		0
7	Lane	4
8	Green	1
9	Samuels	0
10		2
11	Fields	8
12	Nelson	9

Green

Hughes

Kirk

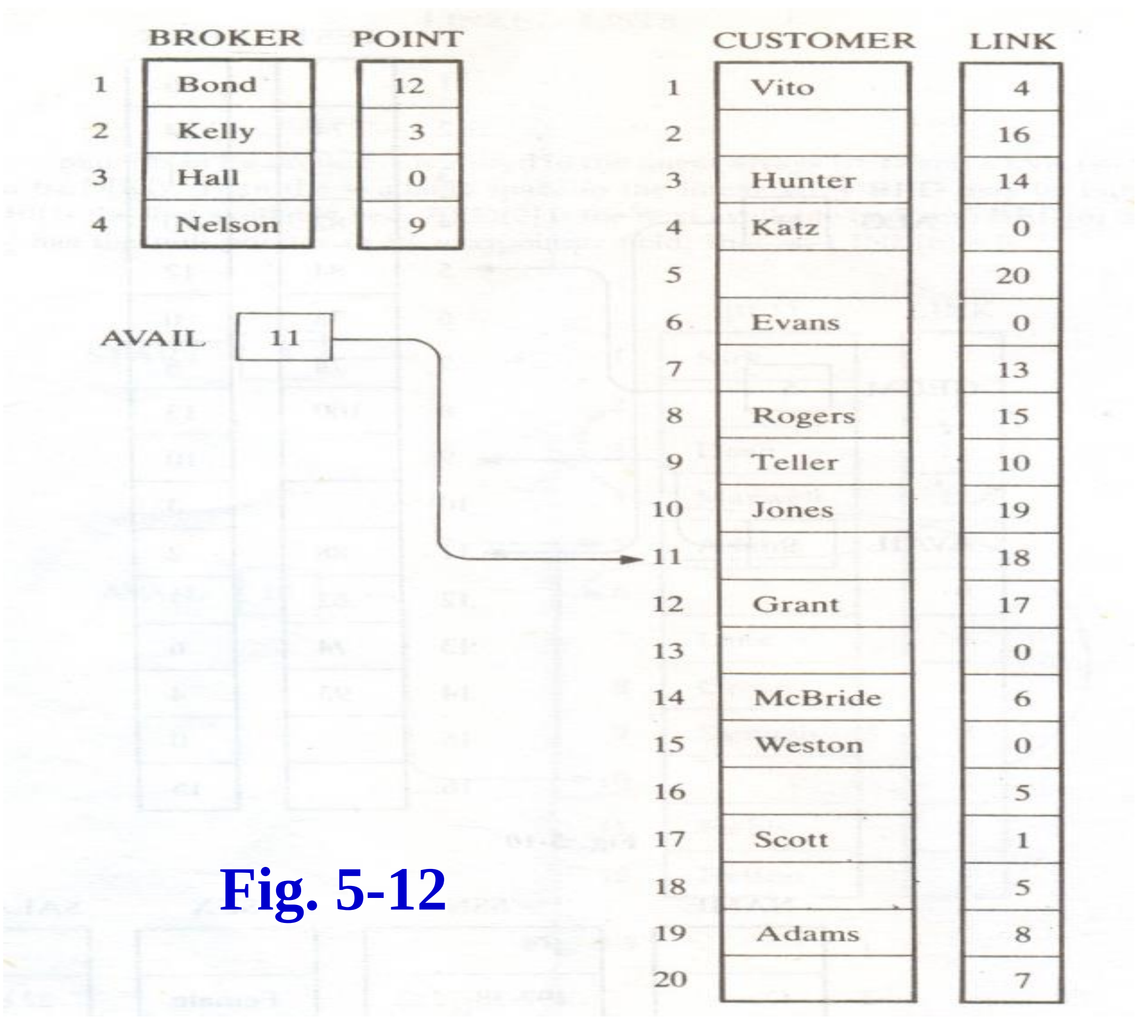


The three changes in the pointer fields follow.

1. LINK[8] = 10. [Now Green points to Hughes.]
2. LINK[10] = 1. [Now Hughes points to Kirk.]
3. AVAIL = 2. [Now AVAIL points to the next available bed.]

EXAMPLE 5.13

(b) Consider Fig. 5-12, not sorted list suppose new will be added at the beginning of the list. Suppose Gordan is a new customer of Kelly



	BROKER	POINT
1	Bond	12
2	Kelly	3
3	Hall	0
4	Nelson	9

AVAIL 11

	CUSTOMER	LINK
1	Vito	4
2		16
3	Hunter	14
4	Katz	0
5		20
6	Evans	0
7		13
8	Rogers	15
9	Teller	10
10	Jones	19
11		18
12	Grant	17
13		0
14	McBride	6
15	Weston	0
16		5
17	Scott	1
18		5
19	Adams	8
20		7

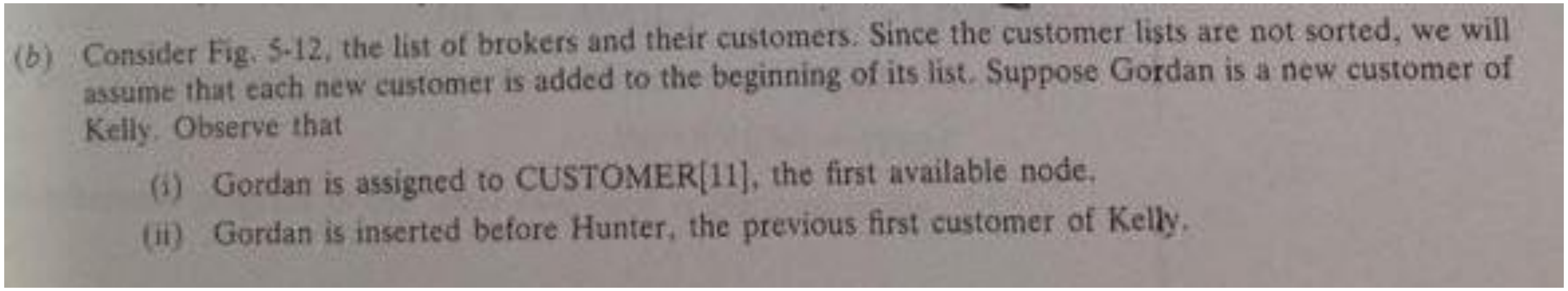
Fig. 5-12

Gordan

Hunter

EXAMPLE 5.13

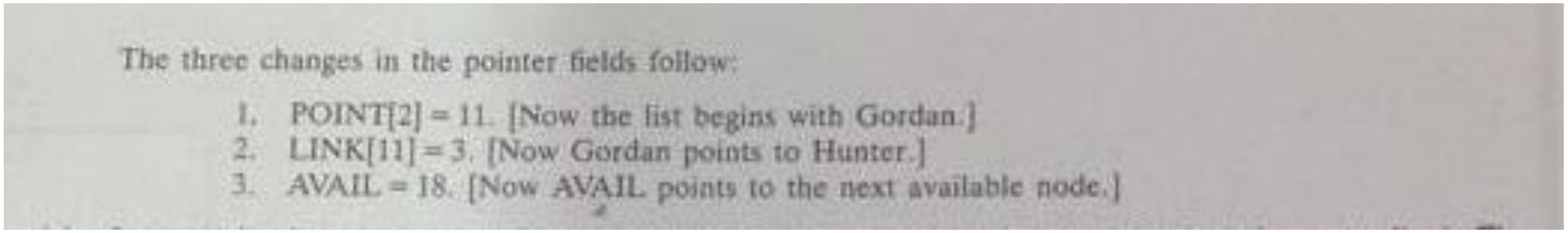
(b) Consider Fig. 5-12, not sorted list suppose new will be added at the beginning of the list. Suppose Gordan is a new customer of Kelly



(b) Consider Fig. 5-12, the list of brokers and their customers. Since the customer lists are not sorted, we will assume that each new customer is added to the beginning of its list. Suppose Gordan is a new customer of Kelly. Observe that

(i) Gordan is assigned to CUSTOMER[11], the first available node.

(ii) Gordan is inserted before Hunter, the previous first customer of Kelly.



The three changes in the pointer fields follow:

1. POINT[2] = 11. [Now the list begins with Gordan.]
2. LINK[11] = 3. [Now Gordan points to Hunter.]
3. AVAIL = 18. [Now AVAIL points to the next available node.]

Gordan

Hunter

EXAMPLE 5.13

(c) Suppose the data elements A, B, C, D, E and F are inserted one after the other into the empty list in Fig 5-13 assume that each new node is inserted at the beginning of the list

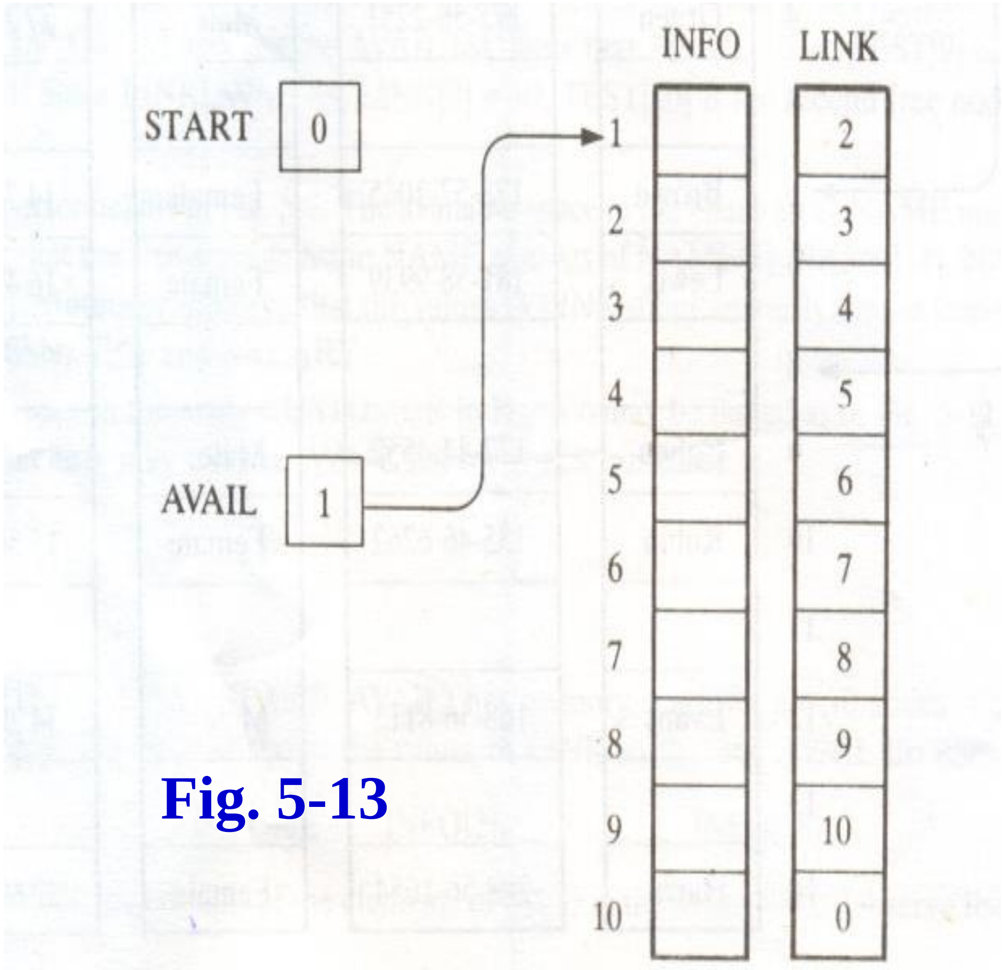
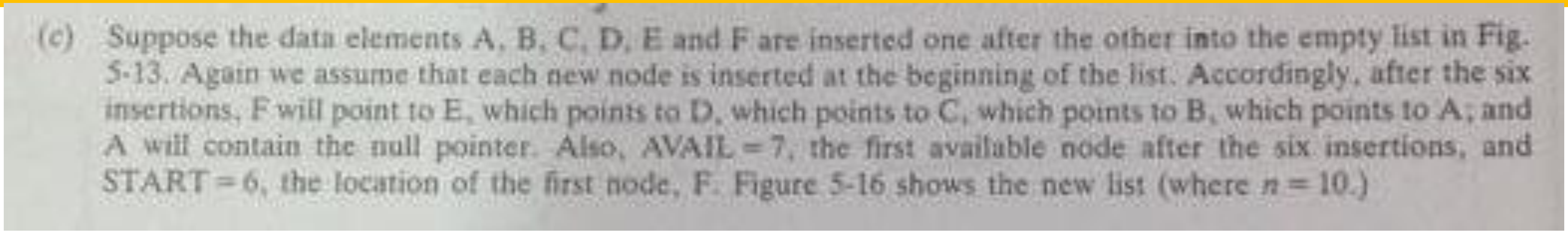
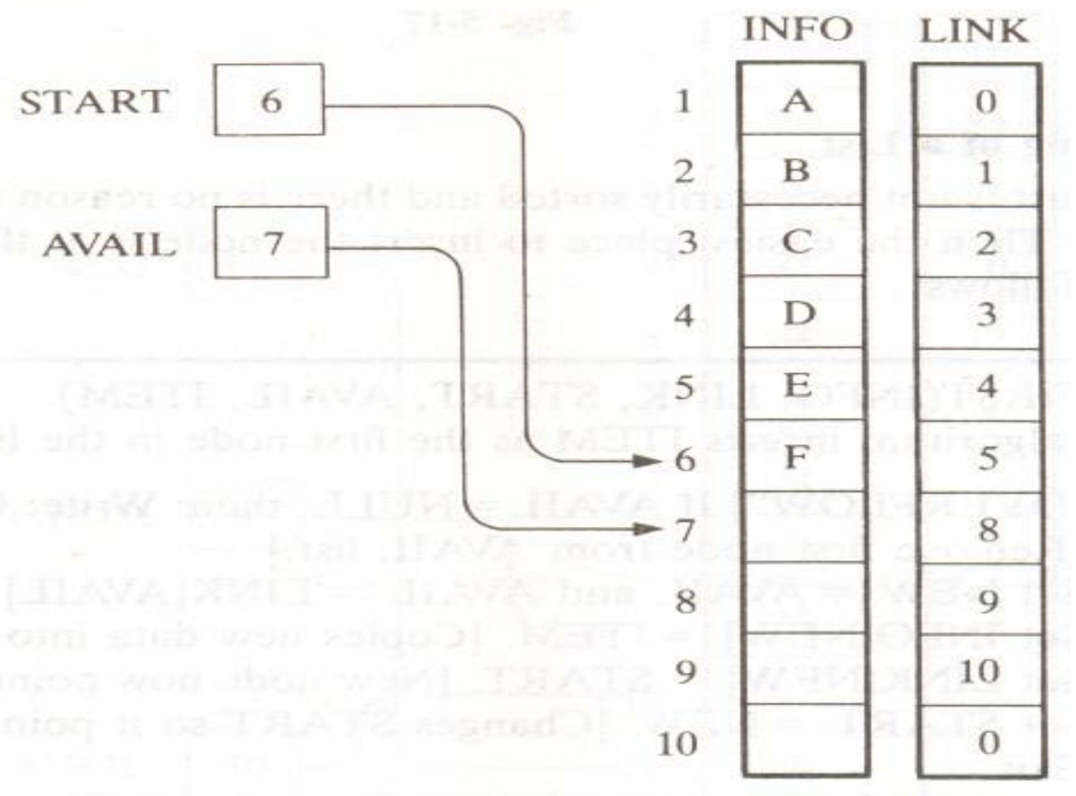


Fig. 5-13

EXAMPLE 5.13



(c) Suppose the data elements A, B, C, D, E and F are inserted one after the other into the empty list in Fig. 5-13. Again we assume that each new node is inserted at the beginning of the list. Accordingly, after the six insertions, F will point to E, which points to D, which points to C, which points to B, which points to A; and A will contain the null pointer. Also, AVAIL = 7, the first available node after the six insertions, and START = 6, the location of the first node, F. Figure 5-16 shows the new list (where $n = 10$.)



	INFO	LINK
1	A	0
2	B	1
3	C	2
4	D	3
5	E	4
6	F	5
7		8
8		9
9		10
10		0

Fig. 5-16

5.7 Inserting a new node

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

a)Check see if space is available in the **AVAIL** list. If not, i.e. if **AVAIL=NULL**, then the algorithm will print the message **OVERFLOW**.

b)Removing the first node from the AVAIL list.

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

c)Copying new information into the new node.

INFO [New] := ITEM

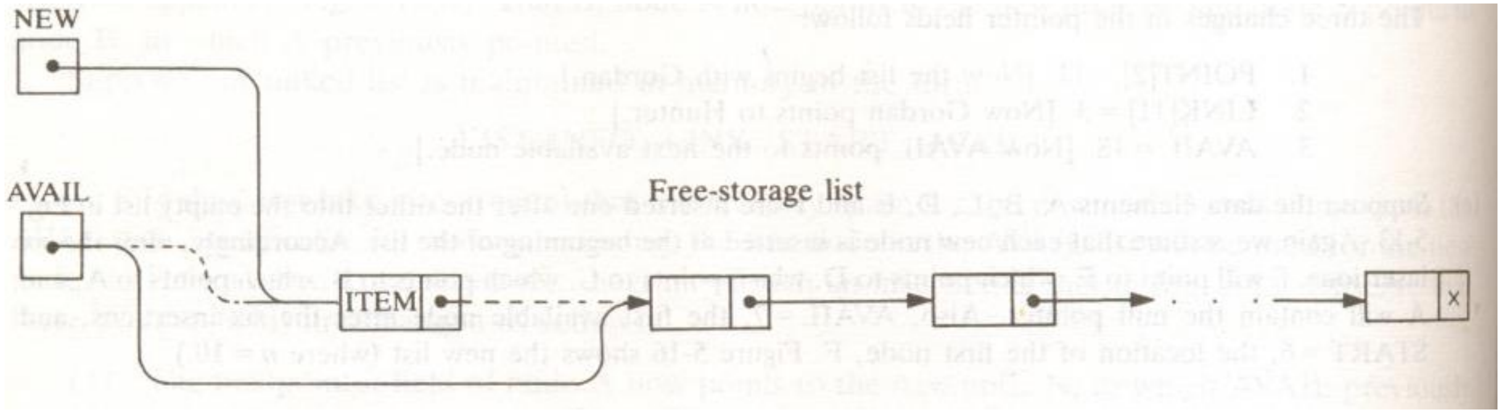


Fig. 5-17

5.7 Inserting a new node

Possible cases of Insert Node

1. Insert into an empty list
2. Insert in front
3. Insert at back
4. Insert in middle

But, in fact, only need to handle two cases:

1. Insert as the first node (Case 1 and Case 2)
2. Insert in the middle or at the end of the list (Case 3 and Case 4)

Insertion Operation

1-Insertion at the Beginning of a List

Algorithm 5.4:

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

START: It has address of first node

AVAIL: It has the address of the first free node in memory

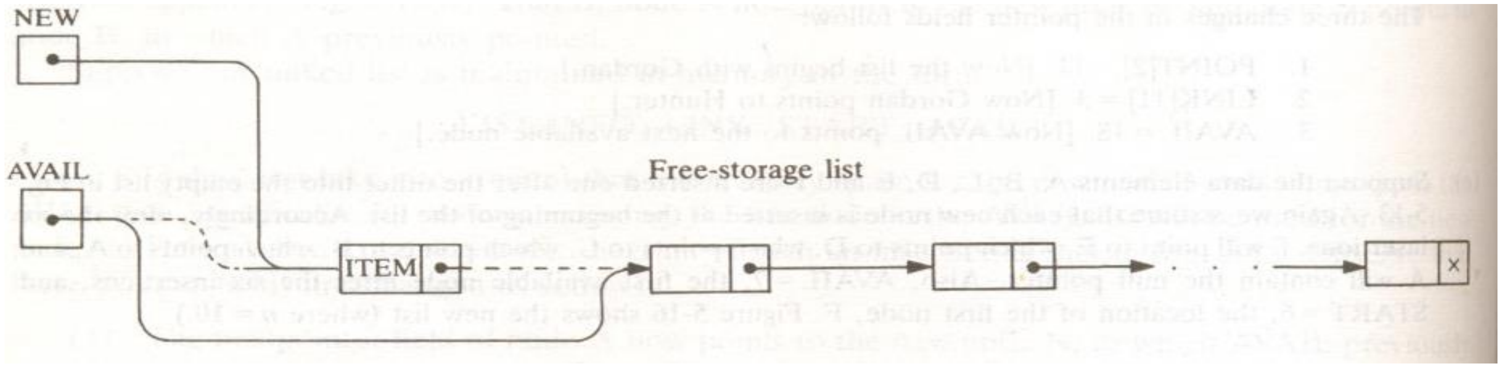
This algorithm inserts ITEM as the first node in the list.

```
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. [Changes START so it points to the new node.]
6.Exit.
```

1-Insertion at the Beginning of a List

Step 1 to 3 have already been discussed.

step 2 and step 3 appears in Fig. 5-17



step 4 and step 5 appears in Fig. 5-18

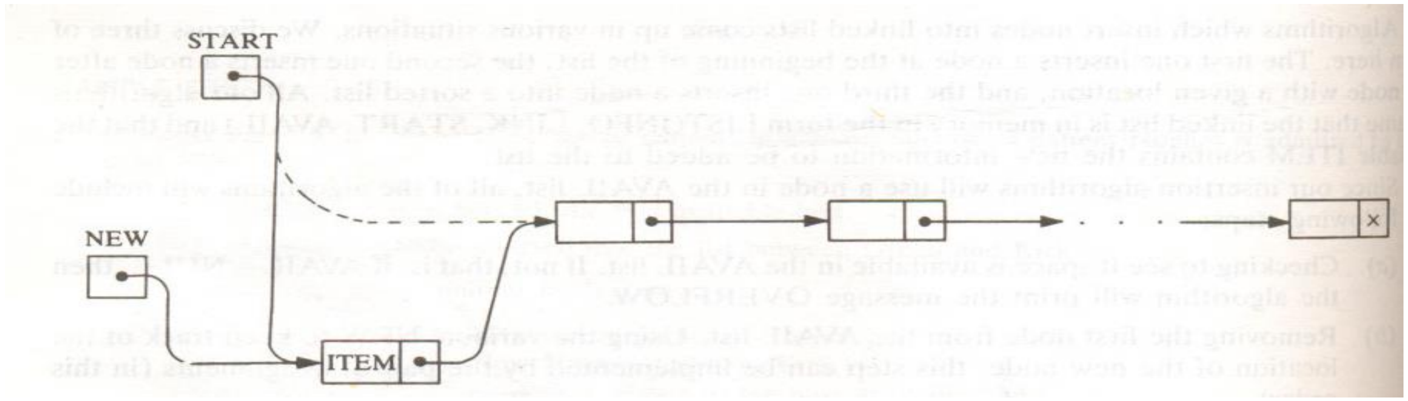
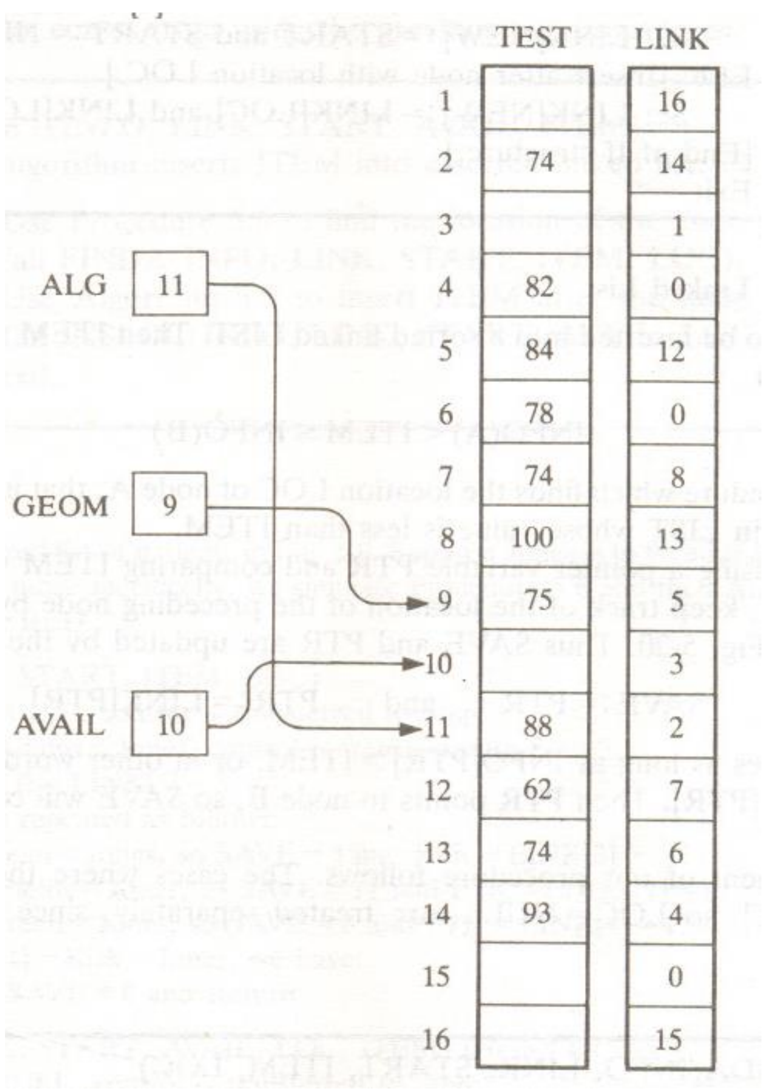


Fig. 5-18 Insertion at the beginning of a list.

Example 5-14



	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

Fig. 5-19

2-Inserting after a given node

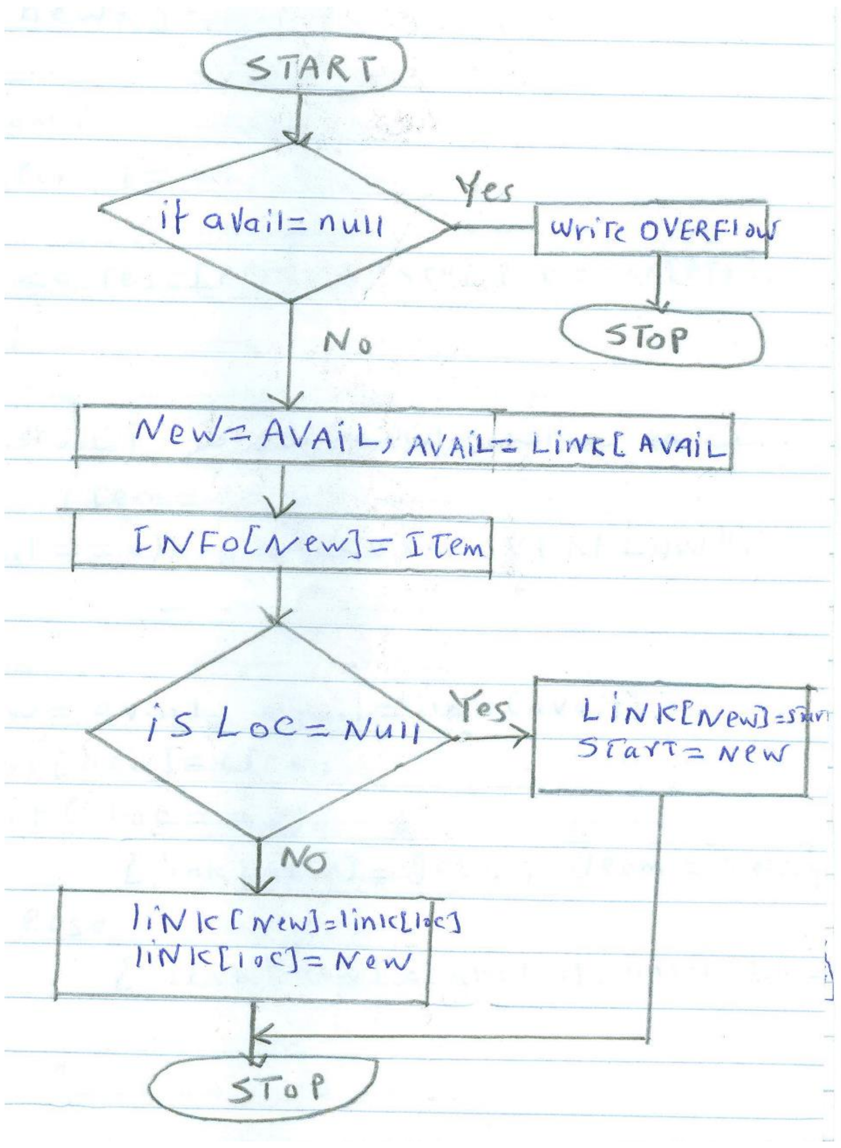
Algorithm 5.5:INSLOC (INFO , LINK , START , AVAIL , LOC , ITEM)

This algorithm inserts ITEM so that ITEM follows the node with location LOC or inserts ITEM as the first node when LOC = NULL.

```
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 of If structure ]
5. Exit.
```

Algorithm 5.5:INSLOC (INFO , LINK , START , AVAIL , LOC , ITEM)

This algorithm inserts ITEM so that ITEM follows the node with location LOC or inserts ITEM as the first node when LOC = NULL.



3-Inserting into a sorted linked list

- ITEM must be inserted between nodes A and B so that

  INFO(A)<ITEM<=INFO(B)
- Traverse the list using a pointer variable PTR
- Comparing the ITEM with INFO[PTR] at each node.
- Keep track the location of the preceding node by a pointer variable SAVE, as in fig 5-20 SAVE and PTR are updated by the assignments

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

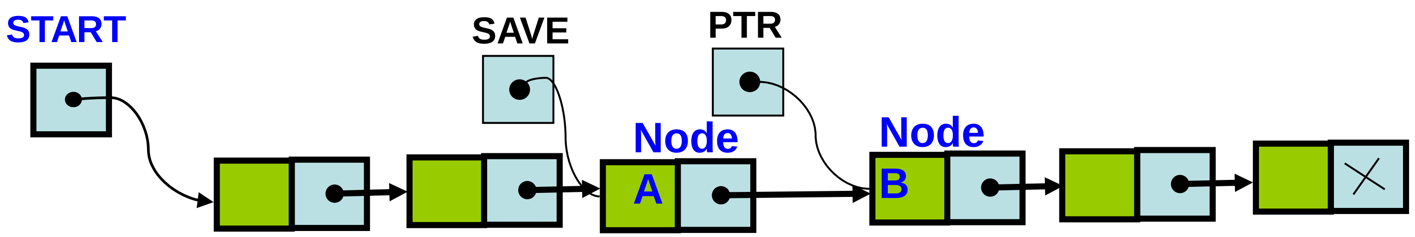


Fig :5-20

Finding a Node

Procedure 5.6: FINDA(INFO, LINK, START, ITEM, LOC)

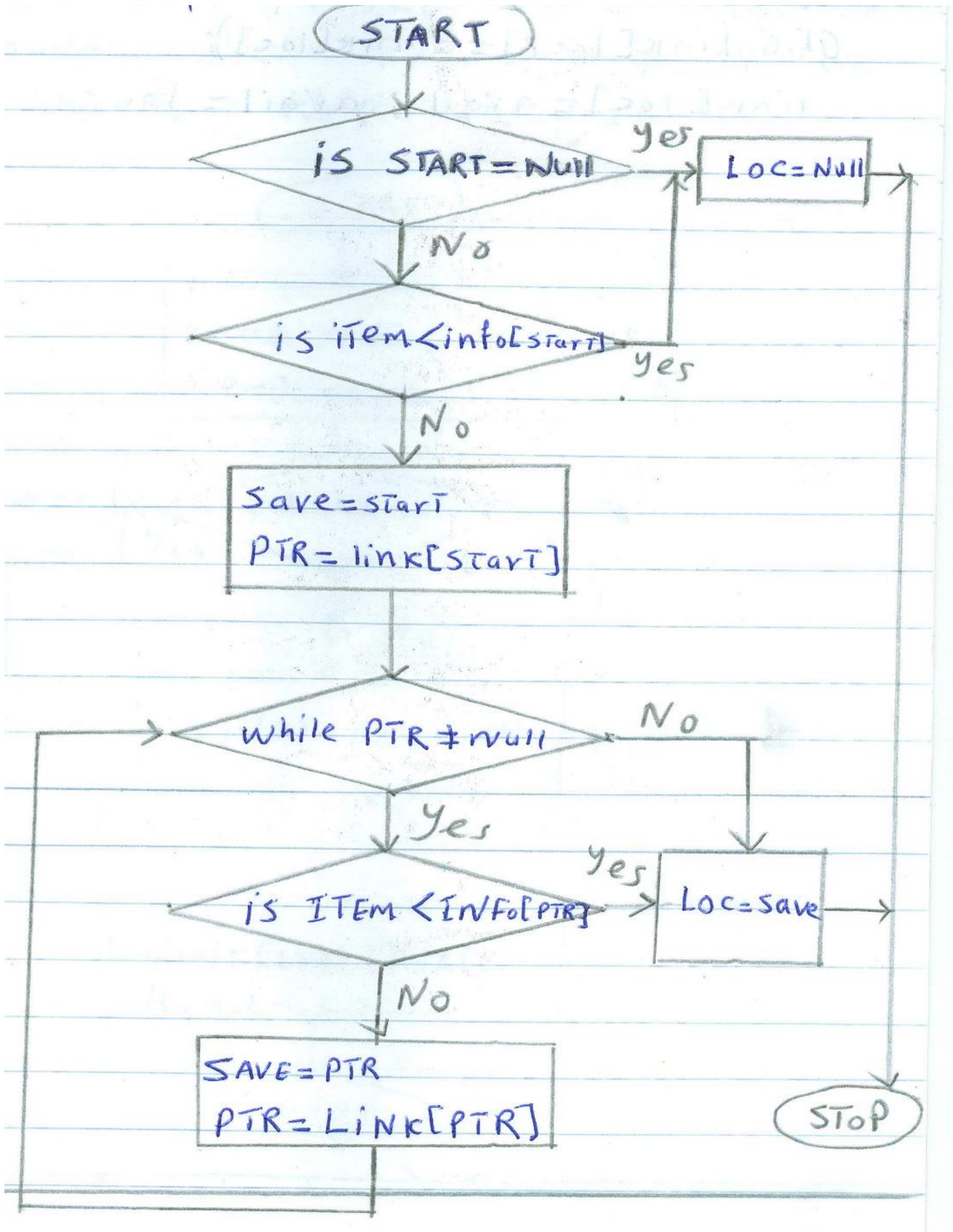
This procedure finds the location LOC of the last node in a sorted list such that

INFO[LOC] < ITEM or sets LOC= NULL

```
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 pointer]
4. Repeat step 5 and 6 while PTR ≠ NULL.
5. If ITEM < INFO[PTR] then:
    Set LOC= SAVE and Return.
6. Set SAVE = PTR and PTR= LINK[PTR] [update pointer]
   (End of step 4)
7. Set LOC= SAVE
8. Return.
```

Procedure 5.6: FINDA(INFO, LINK, START, ITEM, LOC)

This procedure finds the location LOC of the last node in a sorted list such that INFO[LOC] < ITEM or sets LOC= NULL



3-Inserting into a sorted linked list

Algorithm 5.7:

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

This algorithm inserts an item into a sorted linked list.

```
1. [Use Procedure 5.6 to find the location of the node preceding ITEM]
   Call FINDA(INFO, LINK, START, ITEM, LOC).

2. [Use algorithm 5.5 to insert ITEM after the node with location LOC]
   Call INSLOC(INFO, LINK, START, AVAIL, LOC, ITEM).

3. Exit
```

Insertion at the end of the list

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

This procedure adds a new element at the end of LIST.

```
1.    [OVERFLOW?]
      If AVAIL = NULL, then: Write: OVERFLOW and Exit.
2.    [Remove the first node from AVAIL list.]
      Set NEW: = AVAIL, AVAIL : = LINK [AVAIL].
3.    Set INFO [NEW]: = ITEM.            [Copies new data into new node.]
4.    Set LINK [NEW]: = NULL.
5.    If START = NULL, then:
            Set START: = NEW.
      Else:
            Set PTR: = START.                  [Initialize pointer.]
            Repeat Step While LINK [PTR] ≠ NULL:
                  Set PTR: = LINK [PTR]. [End of loop]
            Set LINK [PTR] : = NEW. [End of If structure.]
6.    Exit.
```

Insert after a given position

ADDAFTER (START, LINK, INFO, AVAIL, ITEM, POS)

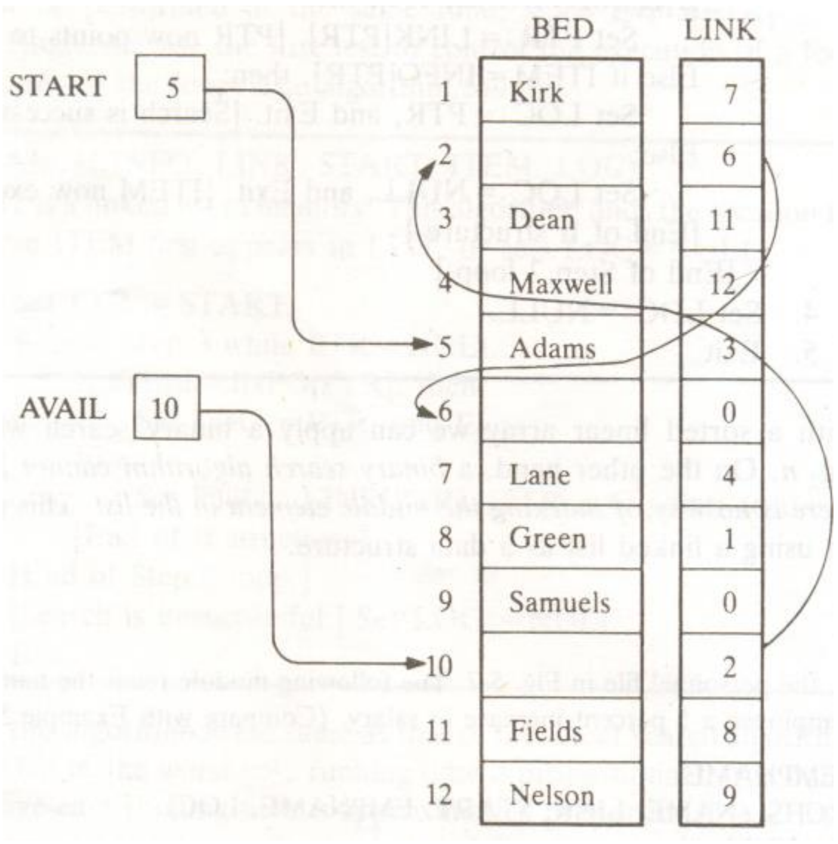
This procedure adds a new element after the given position pos.

```
1.      Set PTR: = START.
2.      Repeat Steps For I: = 1 to POS – 1:
                Set PTR: = LINK [PTR].
        If PTR = NULL, then: Write: "Such a node does not exist " and Exit.
        [End of Step 2 loop.]
3.      [OVERFLOW?]
        If AVAIL = NULL, then: Write: OVERFLOW and Exit.
4.      [Remove the first node from AVAIL list.]
        Set NEW: = AVAIL, AVAIL: = LINK [AVAIL].
5.      Set INFO [NEW]: = ITEM.            [Copies new data into new node.]
6.      Set LINK [NEW]: = LINK [PTR].
7.      Set LINK [PTR]: = NEW.
8.      Exit.
```

EXAMPLE 5.15

Consider Fig. 5-9, the alphabetical list of patients in a ward. Suppose a Jones is to be added

Simulate Algorithm 5.7 =Procedure 5.6 and then Algorithm 5.5

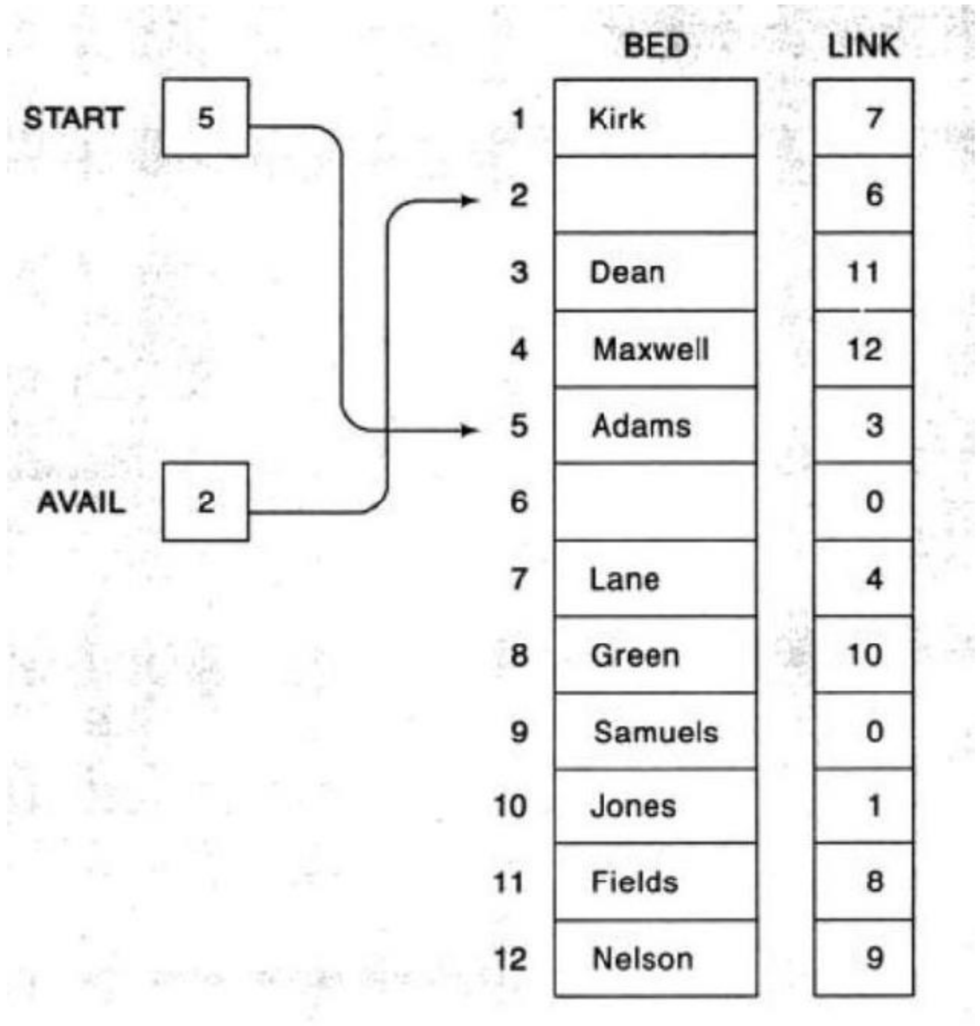


	BED	LINK
1	Kirk	7
2		6
3	Dean	11
4	Maxwell	12
5	Adams	3
6		0
7	Lane	4
8	Green	1
9	Samuels	0
10		2
11	Fields	8
12	Nelson	9

ITEM=Jones

INFO=BED

EXAMPLE 5.15



	BED	LINK
1	Kirk	7
2		6
3	Dean	11
4	Maxwell	12
5	Adams	3
6		0
7	Lane	4
8	Green	10
9	Samuels	0
10	Jones	1
11	Fields	8
12	Nelson	9

Fig. 5-21

Problem

5.6 Suppose NAME1 is a list in memory. Write an algorithm which copies NAME1 into a list NAME2.

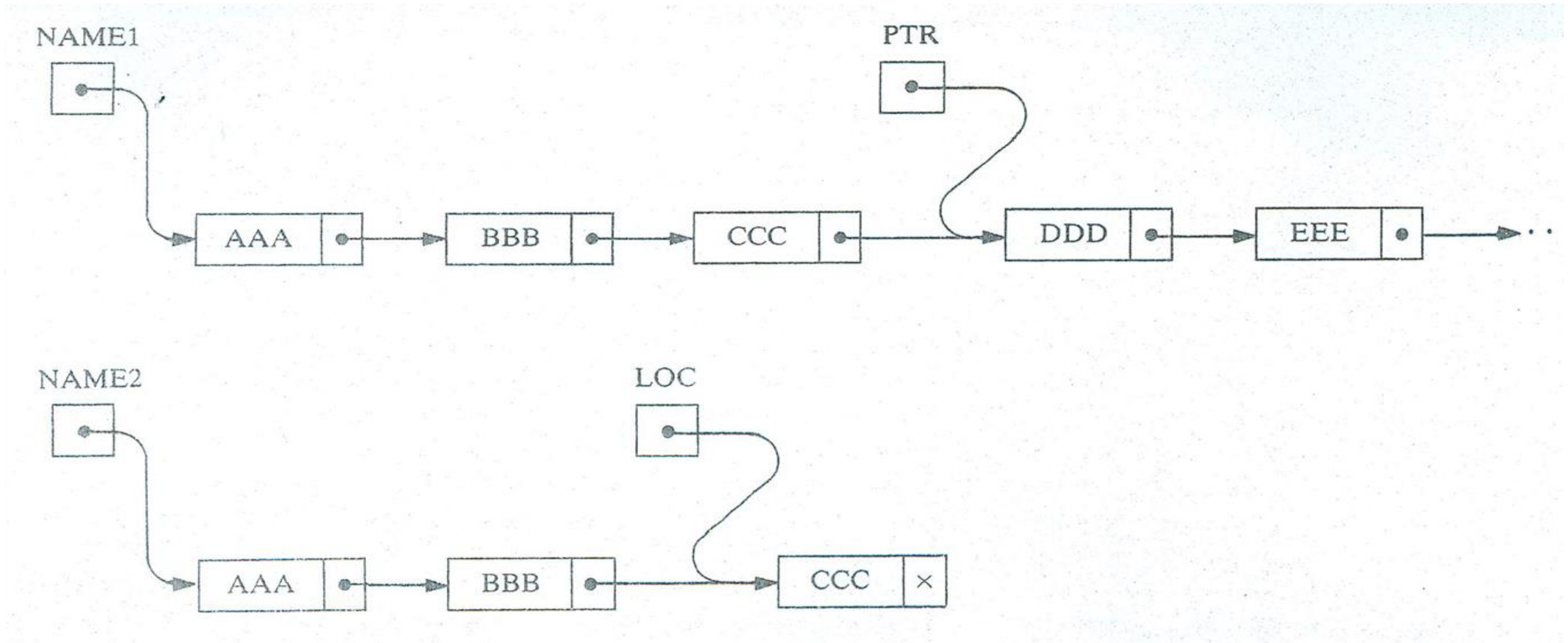


Fig. 5-42

الحذف

Chapter 5: Linked Lists Part III

5.8. Deletion from a linked list

1. Example 5.16
2. Deletion Algorithms
    1. Deleting node following a Given Node
    2. Algorithm 5.8
    3. Deleting node with a Given Item of information
    4. Algorithm 5.9, 5.10 and Example 5.17

5.8 Deletion from a Linked List

5.8 Deletion from a Linked List

- Let LIST be a linked list with a node N between nodes A and B. suppose node N is to be deleted from the linked list.
- The deletion occurs as soon as the next pointer field of node A is changed so that it points to node B.

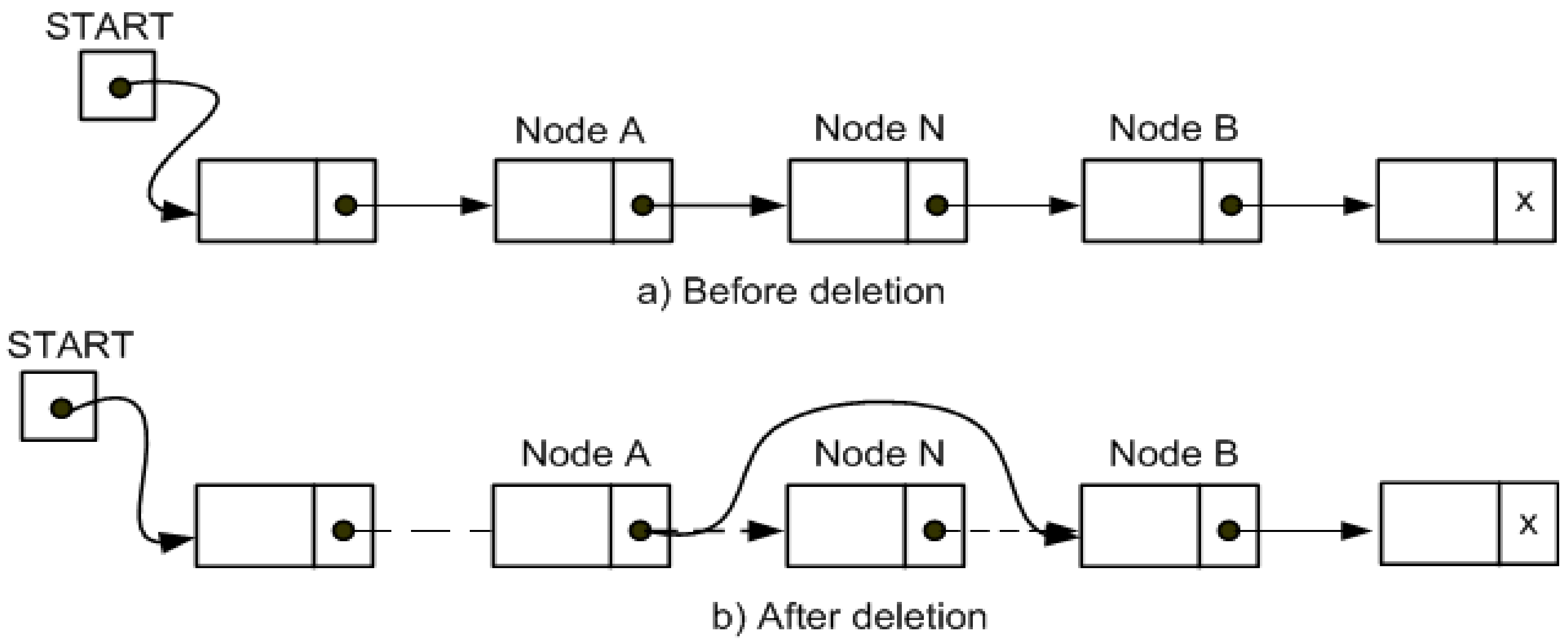


Fig. 5-22

Deletion from a Linked List

- Suppose our linked list is maintained in memory in the form
- LIST(INFO, LINK, START, AVAIL)
- When a node N is deleted from our list, we will immediately return its memory space to the AVAIL list. Specifically, for easier processing, it will be returned to the beginning of the AVAIL list.

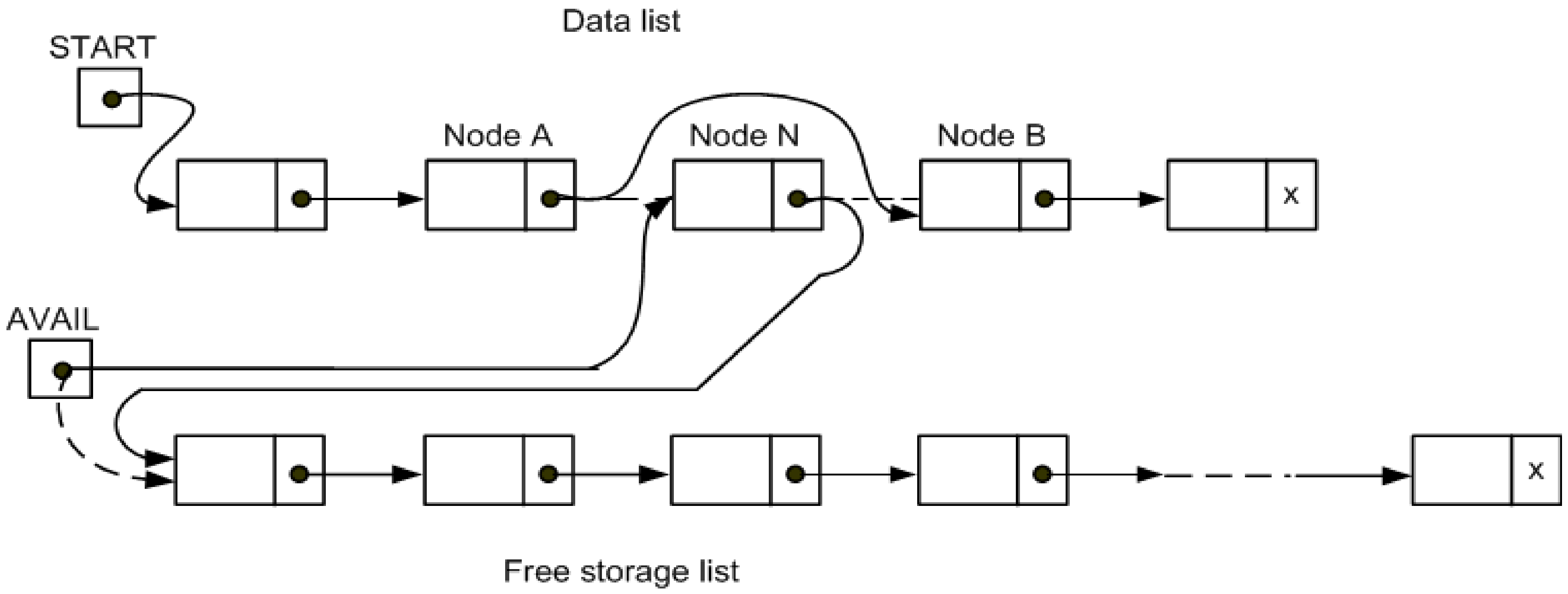


Fig. 5-23

Deletion from a Linked List

The three pointer variables are changed as follows:

1)The nextpointer field of node A now points to node B, where node N previously pointed.

2)The nextpointer field of N now points to the original first node in the free pool, where AVAIL previously pointed.

3)AVAIL now points to the deleted node N.

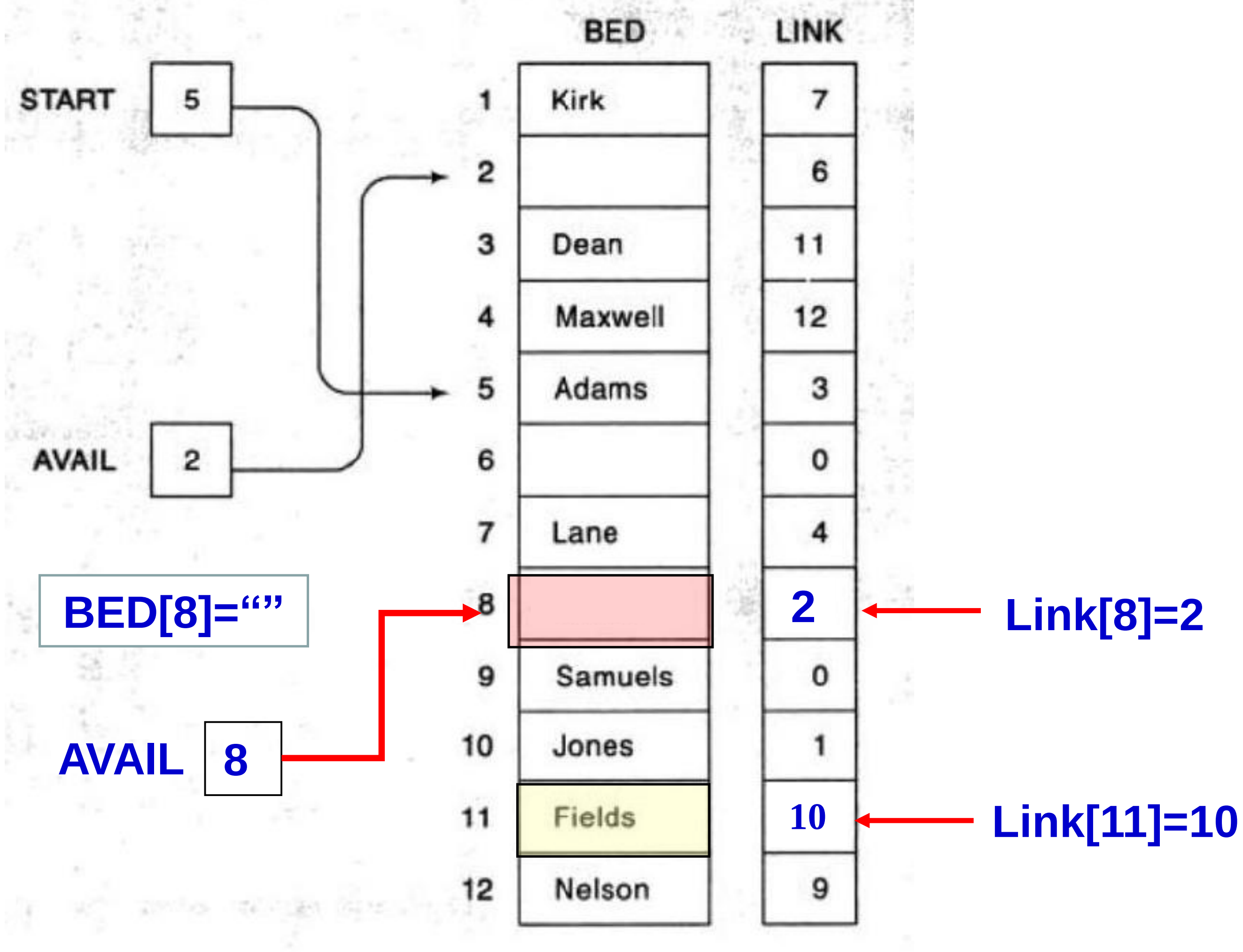
Two special cases

1.If the deleted node **N is the first node** in the list, then START will point to node B; and

2.If the deleted node **N is the last node** in the list, then node A will contain the NULL pointer.

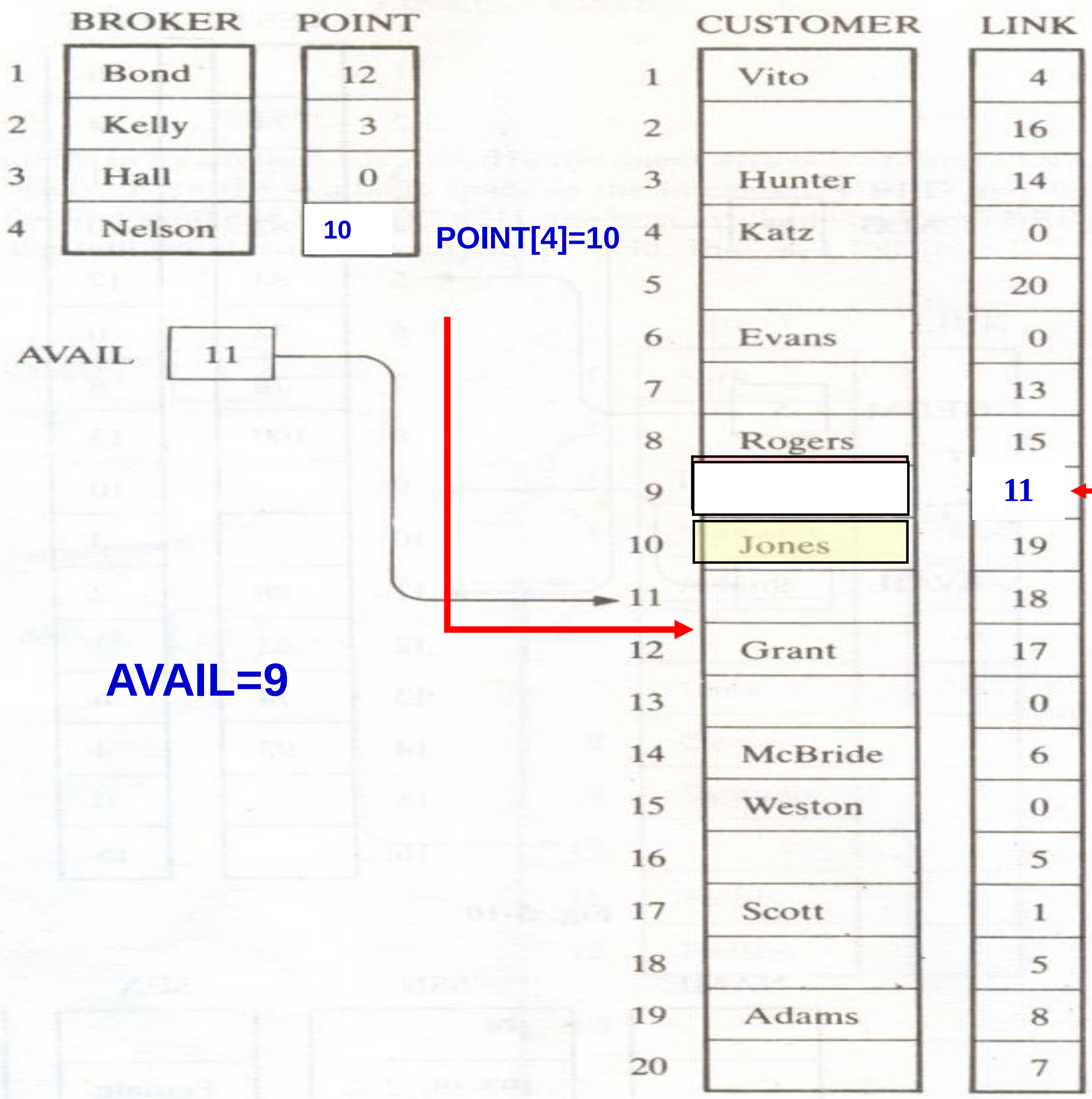
EXAMPLE 5.16 (a)

Green is to be deleted 11 Fields → 8 Green → 10 Jones → 11 Fields → 10 Jones



EXAMPLE 5.16 (b)

special case first element: Teller, the first of Nelson is to be deleted



	BROKER	POINT
1	Bond	12
2	Kelly	3
3	Hall	0
4	Nelson	10

POINT[4]=10

AVAIL 11

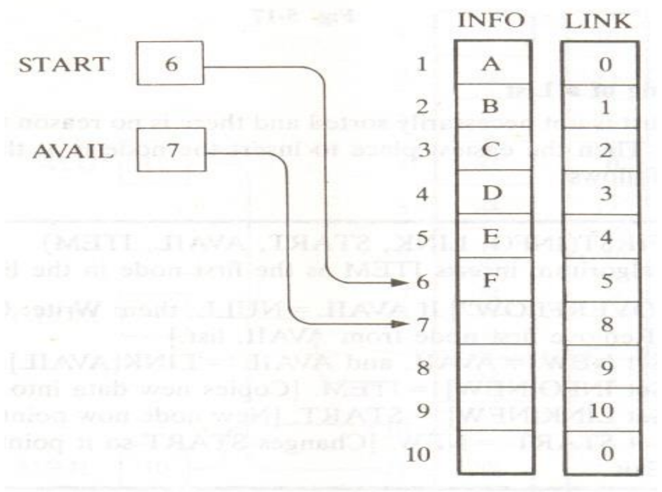
AVAIL=9

	CUSTOMER	LINK
1	Vito	4
2		16
3	Hunter	14
4	Katz	0
5		20
6	Evans	0
7		13
8	Rogers	15
9		11
10	Jones	19
11		18
12	Grant	17
13		0
14	McBride	6
15	Weston	0
16		5
17	Scott	1
18		5
19	Adams	8
20		7

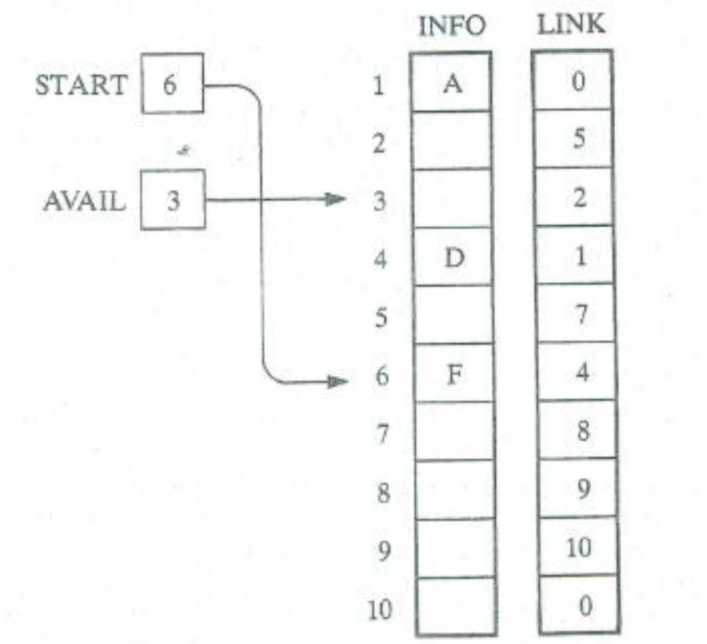
Link[9]=11

EXAMPLE 5.16 (c)

E, B and C are deleted one after the other from the list in Fig. 5-16



	INFO	LINK
1	A	0
2	B	1
3	C	2
4	D	3
5	E	4
6	F	5
7		8
8		9
9		10
10		0



	INFO	LINK
1	A	0
2		5
3		2
4	D	1
5		7
6	F	4
7		8
8		9
9		10
10		0

Fig. 5-24

The first three available nodes are:

INFO[3], which originally contained C
INFO[2], which originally contained B
INFO[5], which originally contained E

1-Deleting the Node Following a Given Node

Deletion Algorithm

Deletion operation deletes a node from the linked list & returns the memory space of the deleted node N to the beginning of the AVAIL list. Accordingly , all algorithms will include the following pair of assignment , where LOC is the location of the deleted node N.

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

These two operation are pictured in following diagram.

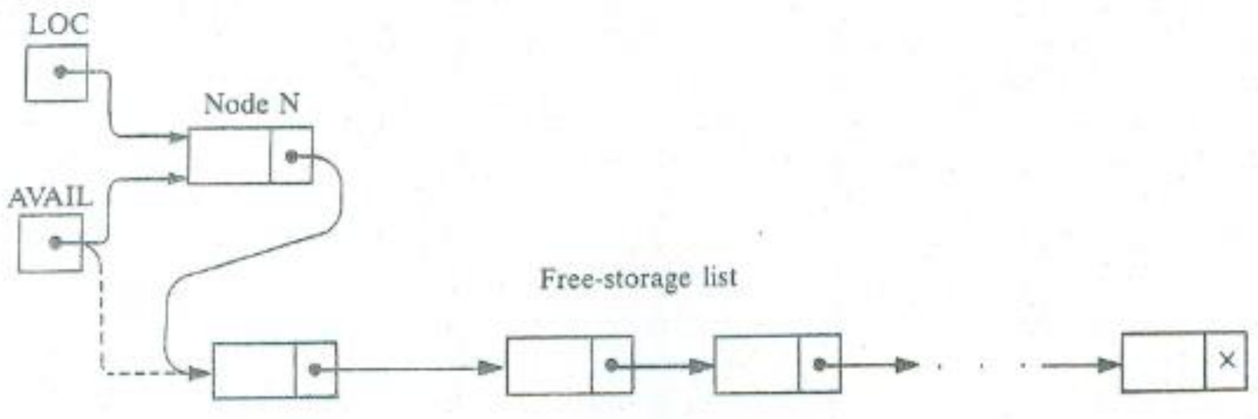


Fig. 5-25 LINK[LOC] := AVAIL and AVAIL := LOC.

Fig. 5-25

1-Deleting the Node Following a Given Node

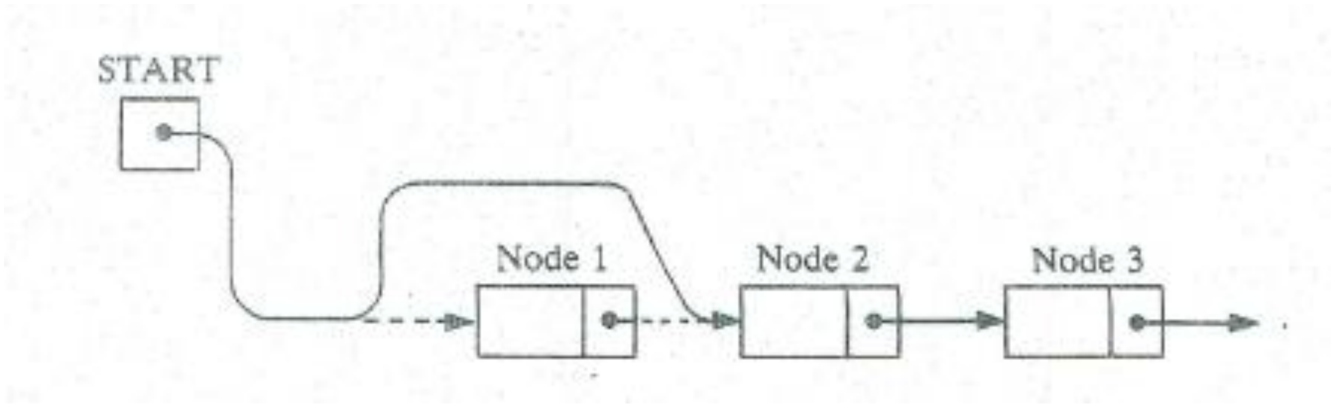


Fig. 5-26 START:=LINK[START]

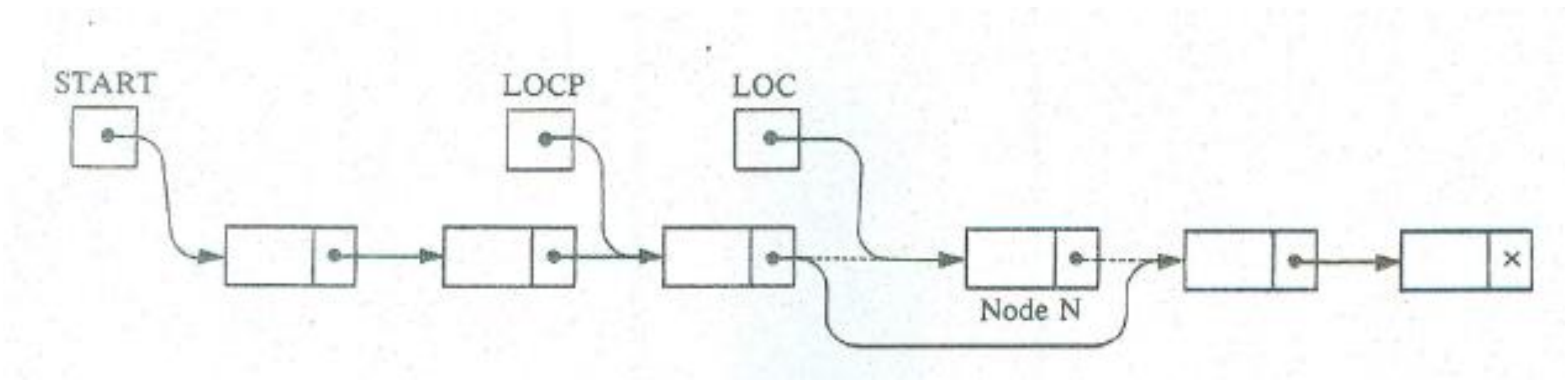


Fig. 5-27 LINK[LOC]:=LINK[LOC]

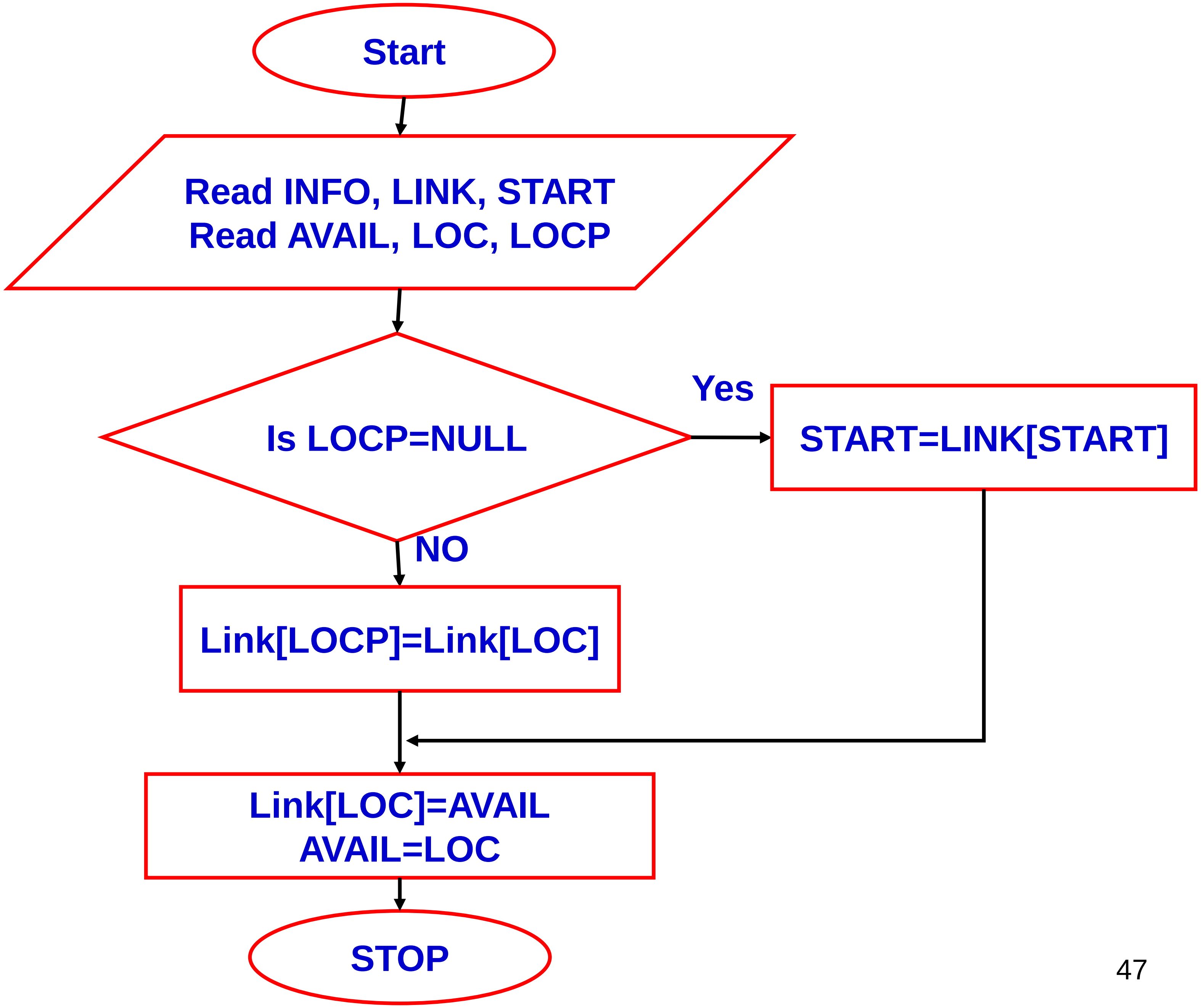
1-Deleting the Node Following a Given Node

Algorithm 5.8.:-

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

This algorithm deletes the node N with location LOC . LOCP is the location of the node which precedes N or , when N is the first node LOCP = NULL.

```
1. If LOCP = NULL , then :
            Set START := LINK [ START ] . [ Deletes first node.]
    Else :
          Set LINK [ LOCP ] := LINK [ LOC ] . [ Deletes node N ]
   [End of If structure .]
2. [ Return deleted node to the AVAIL list .]
        Set LINK [ LOC ] := AVAIL and AVAIL := LOC .
3. Exit
```


Start
Read INFO, LINK, START
Read AVAIL, LOC, LOCP
Is LOCP=NULL
Yes
START=LINK[START]
NO
Link[LOCP]=Link[LOC]
Link[LOC]=AVAIL
AVAIL=LOC
STOP

The algorithm using C#

```
public void del(int[] info, int[] link, int start, int avail, int loc, int locp)
        {
    // Algorithm 5.8: this algorithm deletes the node N with location LOC.
    // LOCP is the location of the node which precedes N or,
    // when N is the first node LOCP =  NULL.

            if(locp==0)
                start = link[start]; //Deletes first node.
            else
                link[locp] = link[loc]; //Deletes node N.

            link[loc] = avail;  //Return deleted node to the AVAIL list
            avail=loc;
        }
```

Problem

Consider the alphabetized list of patients in Fig. 5-9. Determine the changes in the data structure if (*a*) Walters is added to the list and then (*b*) Kirk is deleted from the list.

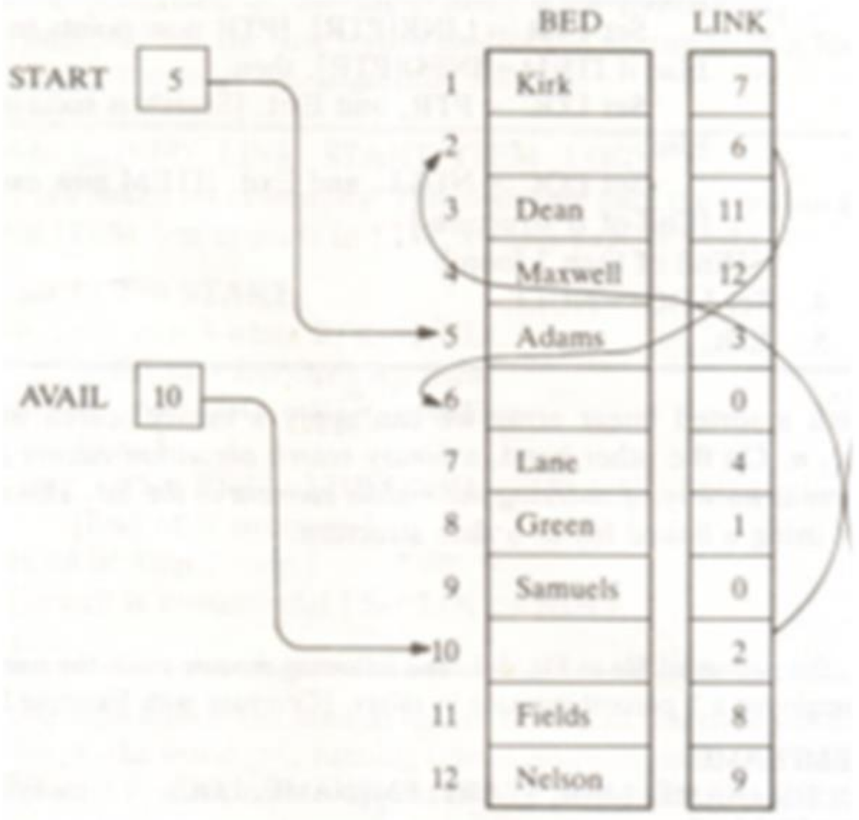
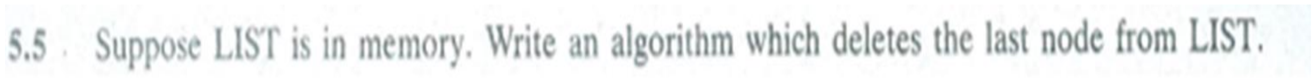


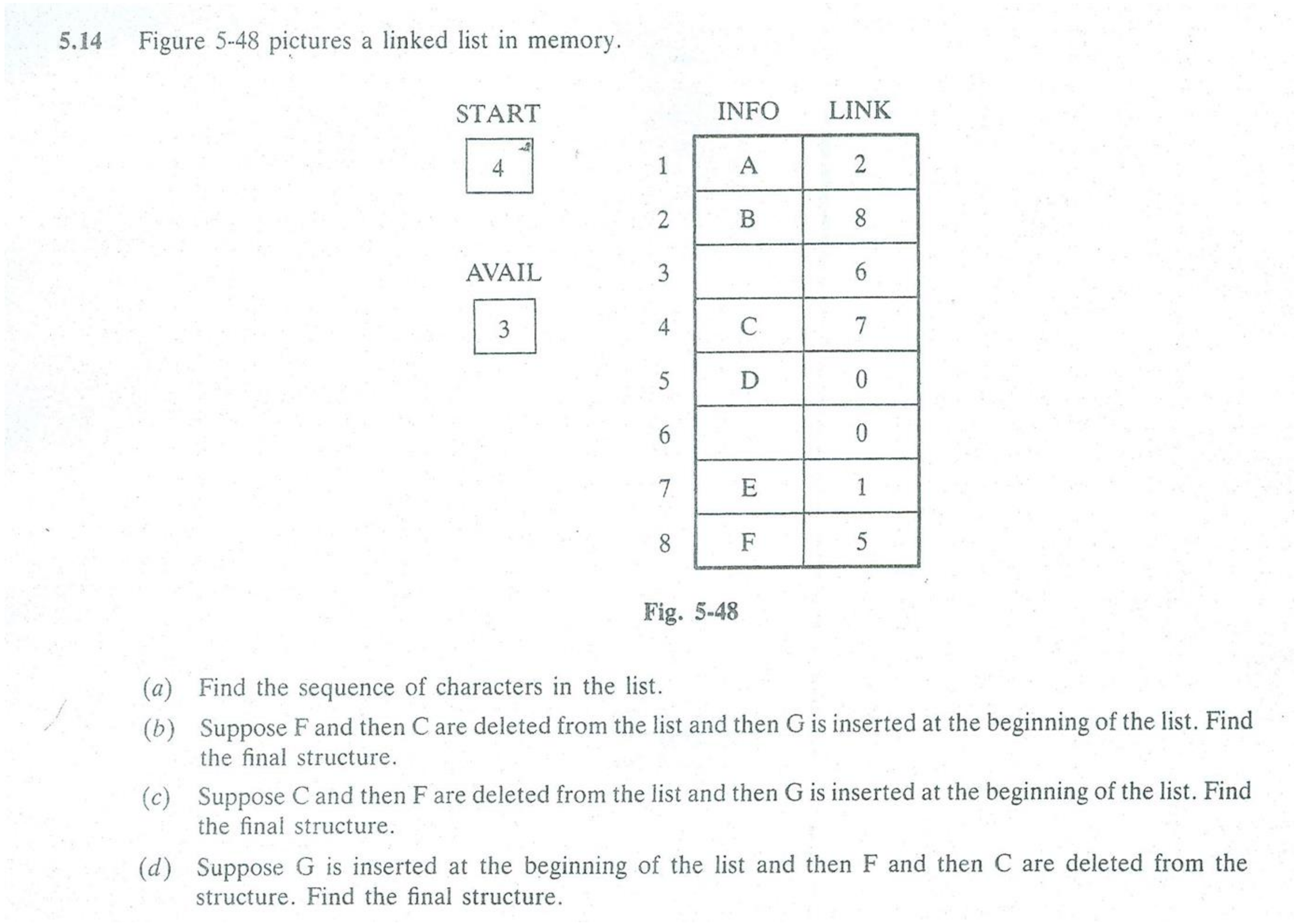
Fig. 5-9

Problem



5.5 Suppose LIST is in memory. Write an algorithm which deletes the last node from LIST.

Problem



5.14 Figure 5-48 pictures a linked list in memory.

START: $\boxed{4}$

AVAIL: $\boxed{3}$

	INFO	LINK
1	A	2
2	B	8
3		6
4	C	7
5	D	0
6		0
7	E	1
8	F	5

Fig. 5-48

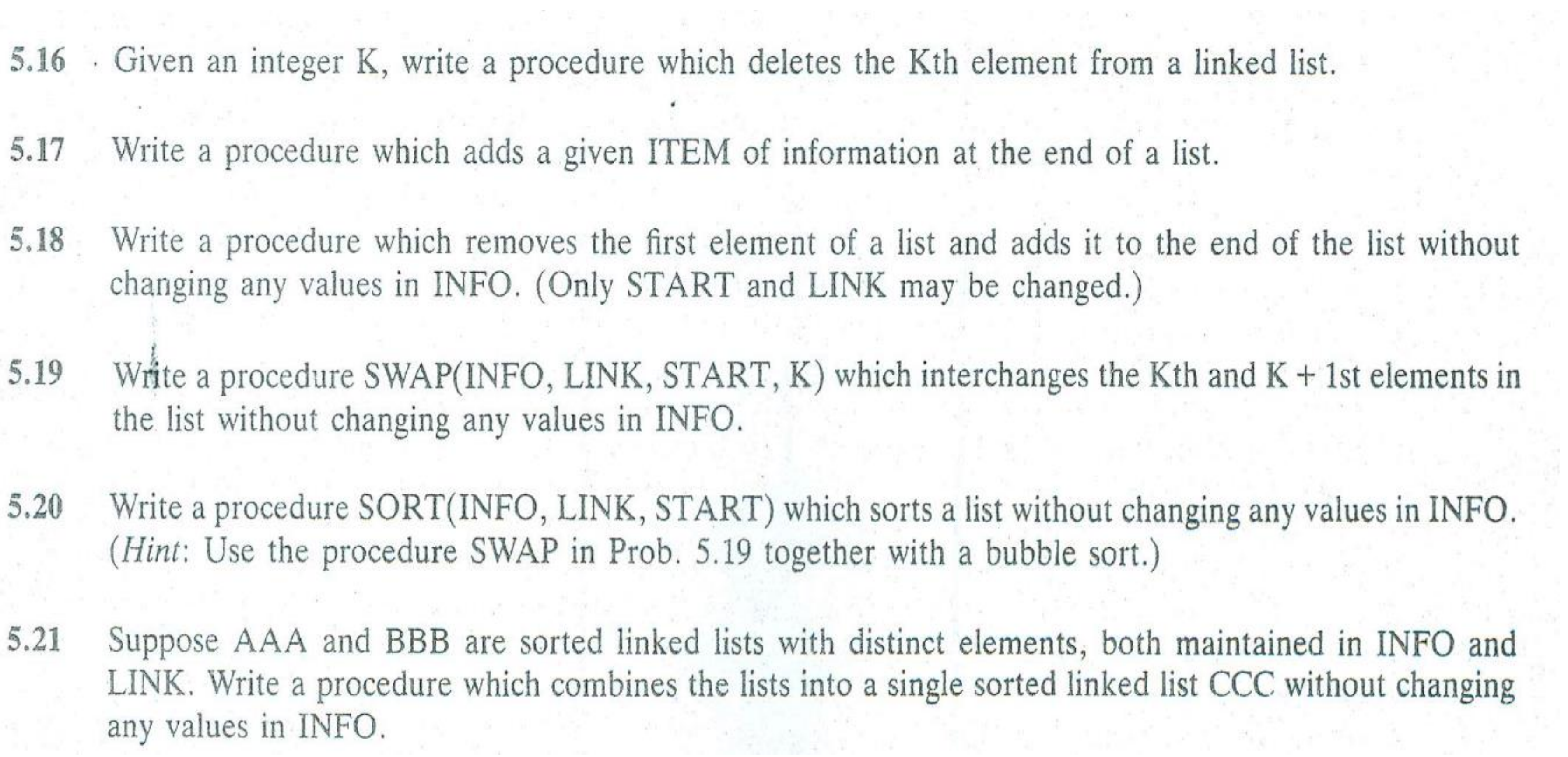
(*a*) Find the sequence of characters in the list.

(*b*) Suppose F and then C are deleted from the list and then G is inserted at the beginning of the list. Find the final structure.

(*c*) Suppose C and then F are deleted from the list and then G is inserted at the beginning of the list. Find the final structure.

(*d*) Suppose G is inserted at the beginning of the list and then F and then C are deleted from the structure. Find the final structure.

Problem



5.16 Given an integer K, write a procedure which deletes the Kth element from a linked list.

5.17 Write a procedure which adds a given ITEM of information at the end of a list.

5.18 Write a procedure which removes the first element of a list and adds it to the end of the list without changing any values in INFO. (Only START and LINK may be changed.)

5.19 Write a procedure SWAP(INFO, LINK, START, K) which interchanges the Kth and K + 1st elements in the list without changing any values in INFO.

5.20 Write a procedure SORT(INFO, LINK, START) which sorts a list without changing any values in INFO. (*Hint*: Use the procedure SWAP in Prob. 5.19 together with a bubble sort.)

5.21 Suppose AAA and BBB are sorted linked lists with distinct elements, both maintained in INFO and LINK. Write a procedure which combines the lists into a single sorted linked list CCC without changing any values in INFO.

2-Deleting the Node with a Given ITEM of Information

Finding a Node

Procedure 5. 9 :

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

This procedure finds the location LOC of the first node N which contains ITEM & the location N.

- If ITEM does not appear in the list , then the procedure sets LOC= NULL ;
- if ITEM appears in the first node then it sets LOCP = NULL.

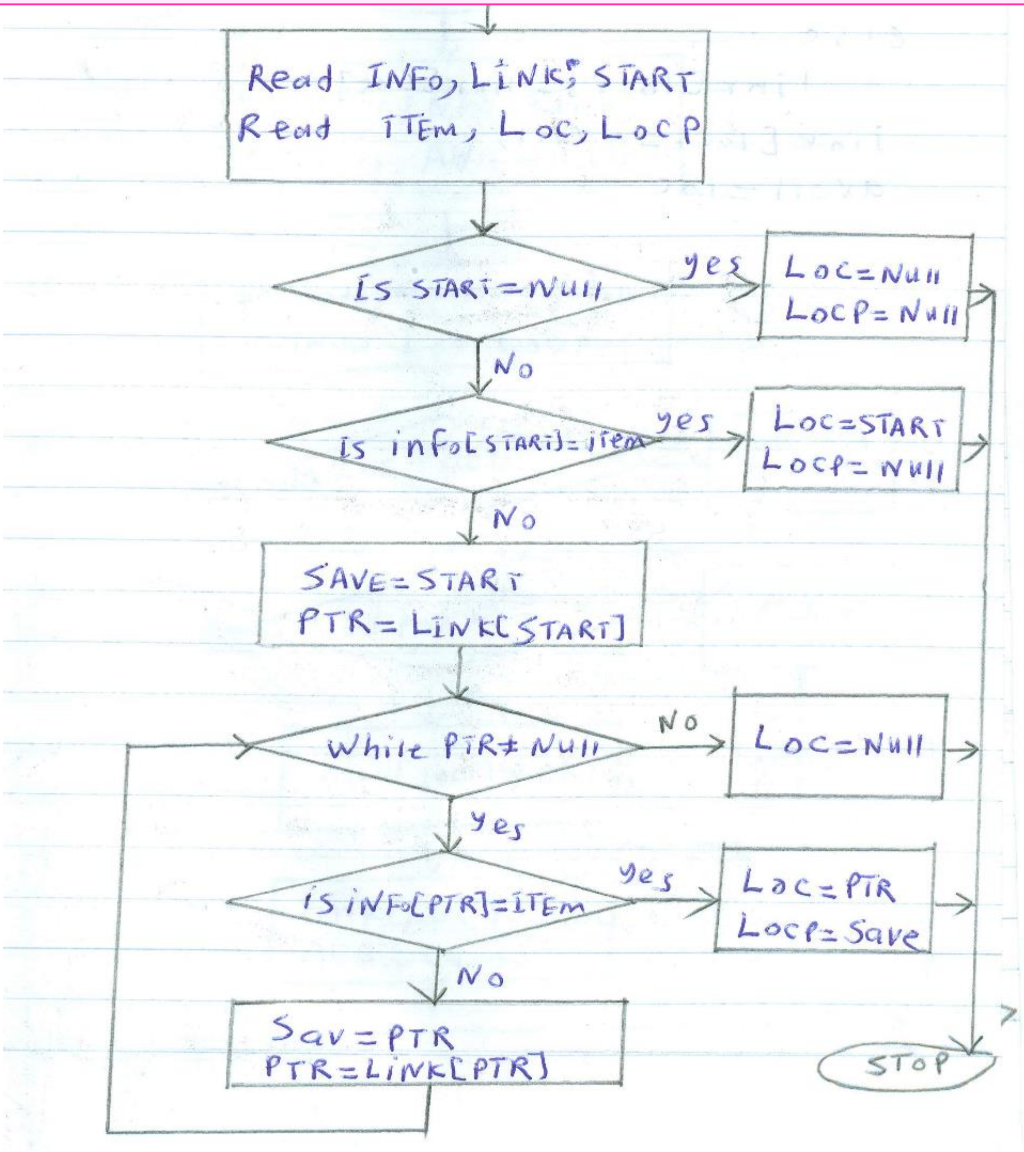
Finding a Node

Procedure 5. 9 :FINDB (INFO, LINK, START, ITEM, LOC, LOCP)

```
1.[ List empty ? ] If START = NULL , then :
        Set  Loc := NULL and LOCP := NULL  , and  Return.
    [End of If structure]
2.[ITEM in first node ?] If  INFO [ START ] := ITEM Then :
        Set LOC := START and LOCP := NULL , and Return.
    [End of If structure]
3. Set SAVE := START and PTR := LINK [ START ] .  [Initiate Pointers.]
4. Repeat steps 5 & 6 while PTR ≠ NULL.
5. If INFO [ PTR ] = ITEM, then :
      Set LOC := PTR and LOCP := SAVE , and Return .
[End of if structure.]
6. Set SAVE := PTR and PTR := LINK [ PTR ] .  [Updates Pointers]
[End of step 4 loop.]
7. Set LOC := NULL [Search Unsuccessful.]
8. Return.
```

Procedure 5. 9 : finds the location LOC of the first node N which contains ITEM & the location LOCP of the node preceding N.

- If ITEM does not appear in the list, then the procedure sets LOC= NULL;
- if ITEM appears in the first node then it sets LOCP = NULL.



```
public int [] findb(int[] info, int[] link, int start, int item)
    {
        // Procedure 5.9: find LOC and LOCP
        int loc, locp,save,ptr;
       int[] output = new int[2];
        if(start==0) //Is list empty?
        {
            loc=0; locp=0;
            output[0] = loc; output[1] = locp; return output;
        }
        if(info[start]==item) //is ITEM in first node?
        {
            loc=start; locp=0;
            output[0] = loc; output[1] = locp; return output;
        }
        save=start; ptr=link[start]; //Initiate Pointers.
        while(ptr !=0)
        {
            if(info[ptr]==item)
            {
                loc=ptr; locp=save;
                output[0] = loc; output[1] = locp; return output;
            }
            save=ptr; ptr=link[ptr]; //Updates Pointers.
        }
        loc=0; locp=0;//Search Unsuccessful.
        output[0] = loc; output[1] = locp; return output;
    }
```

2-Deleting the Node with a Given ITEM of Information

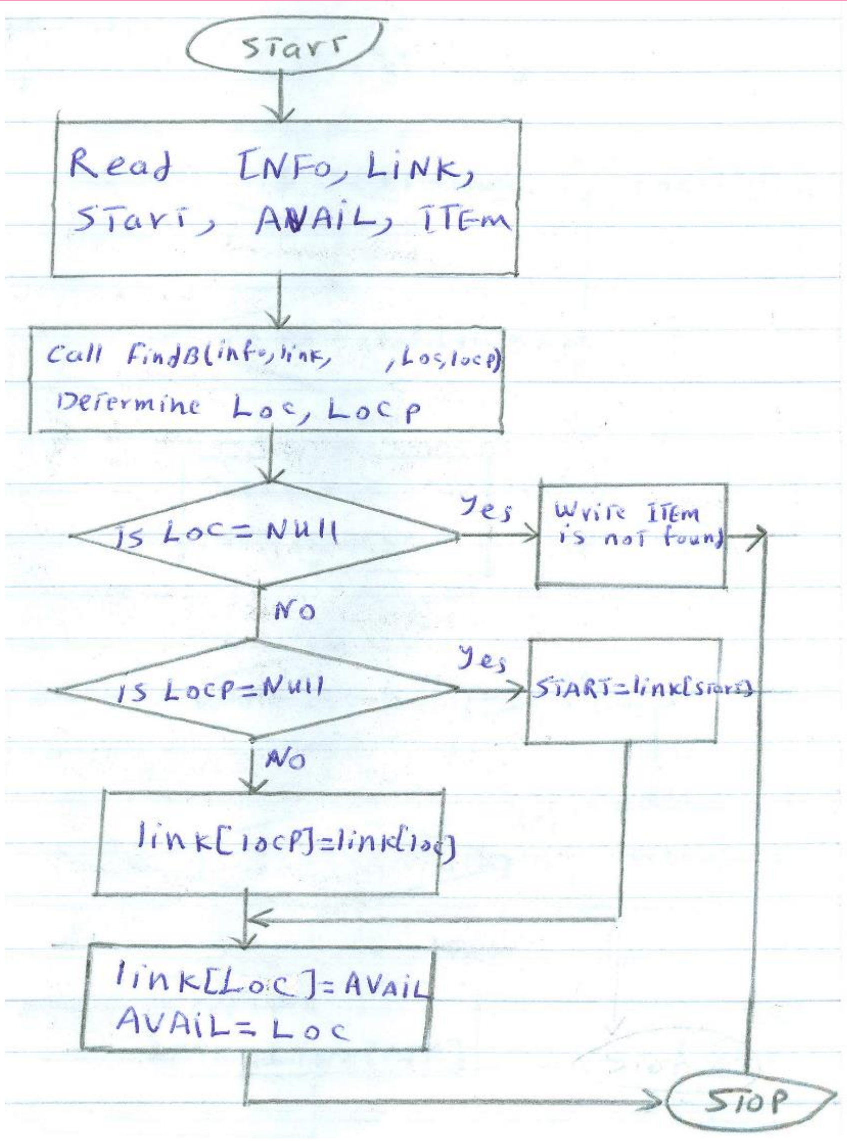
Algorithm 5.10 :- DELETE (INFO, LINK , START ,AVAIL , ITEM)

This algorithm deletes from a linked list the first node N which contains the given ITEM of information.

```
1. Call FINDB (INFO, LINK, START, ITEM, LOC, LOCP)
([use procedure 5.9 to find the location of N & its preceding node.]
2. If  LOC = NULL Then:  Write : ITEM not in list and Exit.
3.  Call DEL(INFO,LINK,START,AVAIL,LOC,LOCP)
4. Exit.
```

Algorithm 5.10 :- DELETE (INFO, LINK , START ,AVAIL , ITEM)

This algorithm deletes from a linked list the first node N which contains the given ITEM of information.

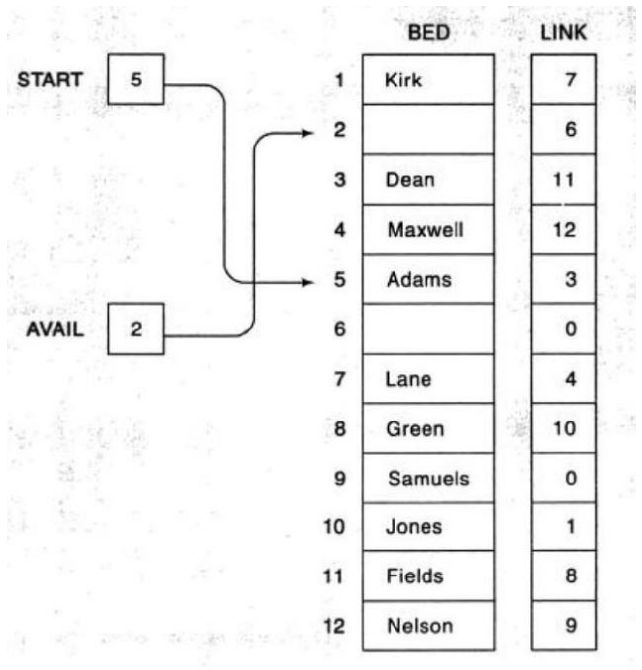


Start
Read INFO, LINK, Start, AVAIL, ITEM
Call FindB(info, link, , Loc, locP) Determine Loc, LocP
is Loc = Null
Yes
Write ITEM is not found
No
is LocP = Null
Yes
START = link[start]
No
link[locP] = link[loc]
link[Loc] = AVAIL AVAIL = Loc
STOP

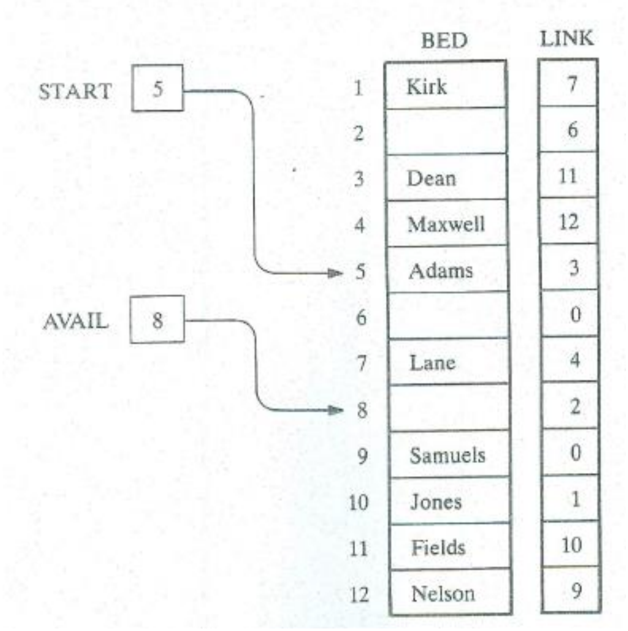
EXAMPLE 5.17

- **Green is to be discharged**
- **Simulate Procedure 5.9 to find the location LOC of Green and the LOCP of the patient proceeding Green.**
- **Then simulate Algorithm 5.10 to delete Green**

ITEM=Green, INFO=BED, START=5 and AVAIL=2



	BED	LINK
1	Kirk	7
2		6
3	Dean	11
4	Maxwell	12
5	Adams	3
6		0
7	Lane	4
8	Green	10
9	Samuels	0
10	Jones	1
11	Fields	8
12	Nelson	9



	BED	LINK
1	Kirk	7
2		6
3	Dean	11
4	Maxwell	12
5	Adams	3
6		0
7	Lane	4
8		2
9	Samuels	0
10	Jones	1
11	Fields	10
12	Nelson	9

Fig. 5-28

Sorting A Linked list

SORT (INFO, LINK, START)

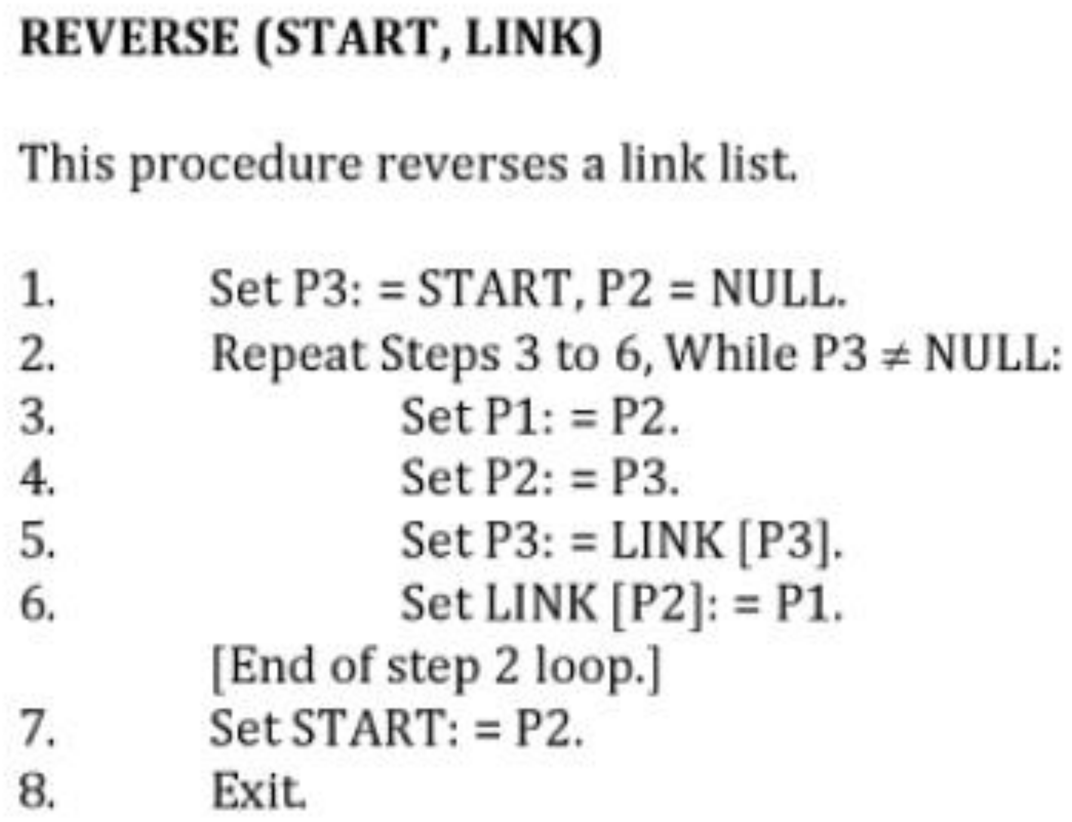
This procedure sorts a list using the selection sorting technique.

```
1.    Set P1: = START.
2.    Repeat Steps 3 to 5 while P1 ≠ NULL:
3.          Set P2: = LINK [P1].
4.          Repeat Step While P2 ≠ NULL:
                  If INFO [P1] > INFO [P2] , then:
                        (a) Set T: = INFO [P1].
                        (b) Set INFO [P1]: = INFO [P2].
                        (c) Set INFO [P2]: = T.
                  [End of If Structure.]
                  Set P2: = LINK [P2].
            [End of Step 4 loop.]
5.          Set P1: = LINK [P1].
      [End of step 2 loop.]
6.    Exit.
```

Sorting A Linked list

```
public void selection_sort(int[] info, int[] link, int start)
        {
            int p1,p2,temp;
            p1 = start ;
            while(p1 !=0)
                {
                p2 = link[p1] ;
                while(p2 !=0)
                {
                    if (info[p1] > info[p2] )
                    {
                        temp = info[p1];
                        info[p1] = info[p2] ;
                        info[p2] = temp ;
                    }
                    p2= link[p2];
                }
                p1 = link[p1] ;
            }
        }
```

Reverses a Linked List



REVERSE (START, LINK)

This procedure reverses a link list.

```
1.    Set P3: = START, P2 = NULL.
2.    Repeat Steps 3 to 6, While P3 ≠ NULL:
3.        Set P1: = P2.
4.        Set P2: = P3.
5.        Set P3: = LINK [P3].
6.        Set LINK [P2]: = P1.
      [End of step 2 loop.]
7.    Set START: = P2.
8.    Exit.
```

Reverses a Linked List

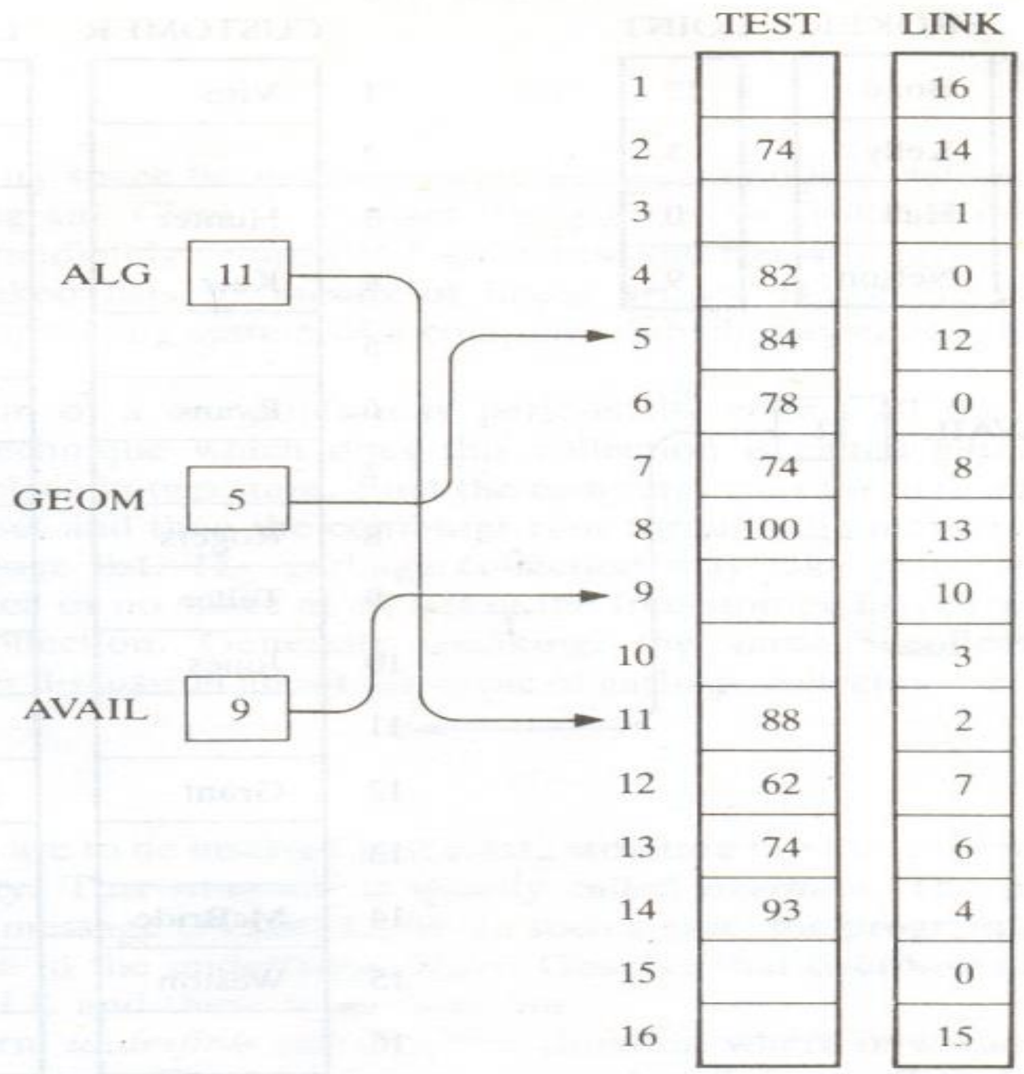
```
public int reverse(int[] info, int[] link, int start)
        {
            int p1, p2, p3;
            p3 = start;
            p2 = 0;
            while (p3 != 0)
            {
                p1 = p2;
                p2 = p3;
                p3 = link[p3];
                link[p2]=p1;
            }
            start = p2;
            return start;
        }
```

Deleting a node

- **DELETE(INFO, LINK, START, AVAIL, ITEM)**
  - **Delete a node with the value equal to ITEM from the list.**
  - **If such a node is found, return its position. Otherwise, return NULL.**
- **Steps**
  - Find the desirable node (similar to **FINDB**)
  - Release the memory occupied by the found node
  - Set the pointer of the predecessor of the found node to the successor of the found node
- Like **INSLOC**, there are two special cases
  - Delete first node
  - Delete the node in middle or at the end of the list

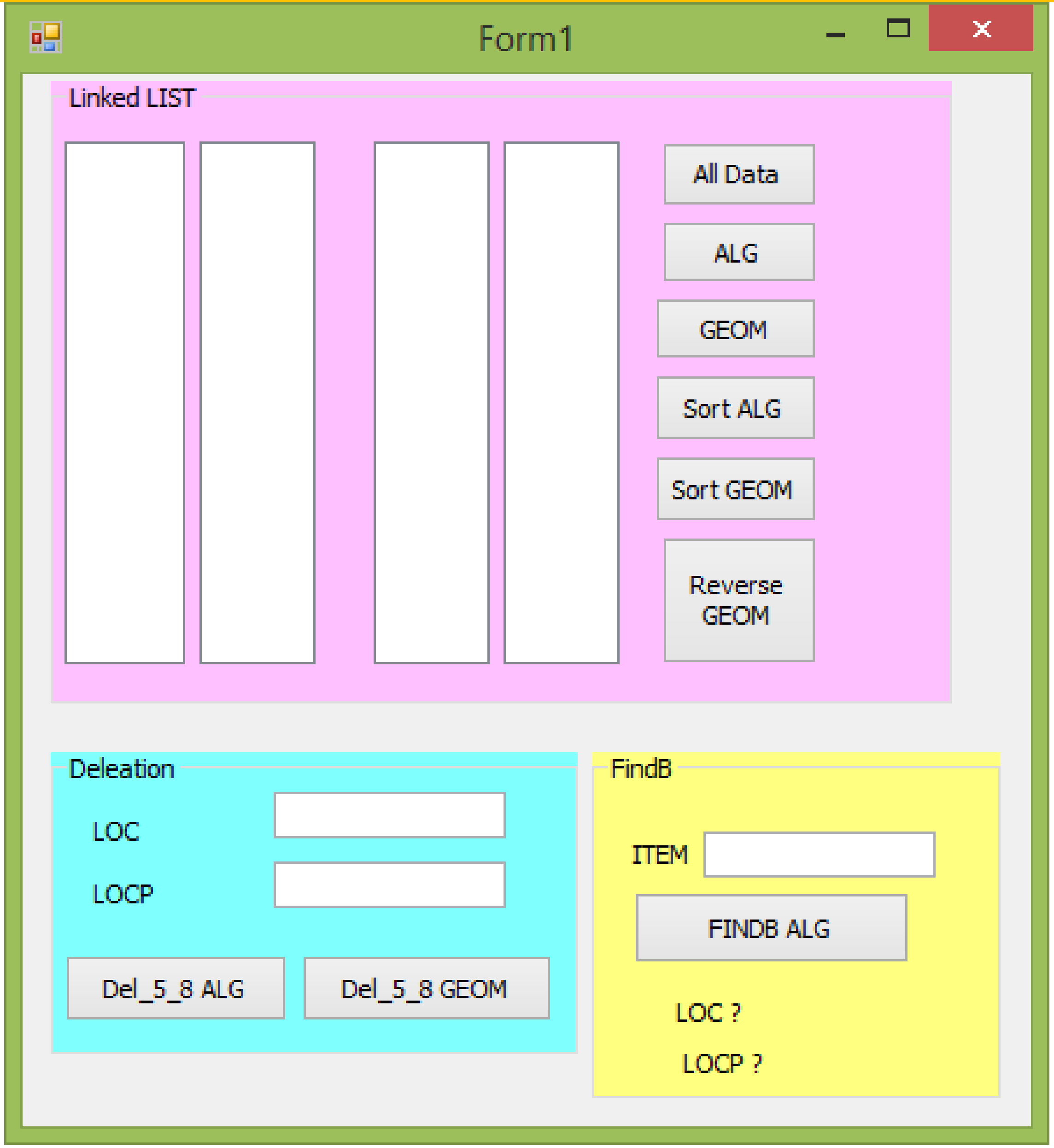
Program using C#

Program using C#



	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

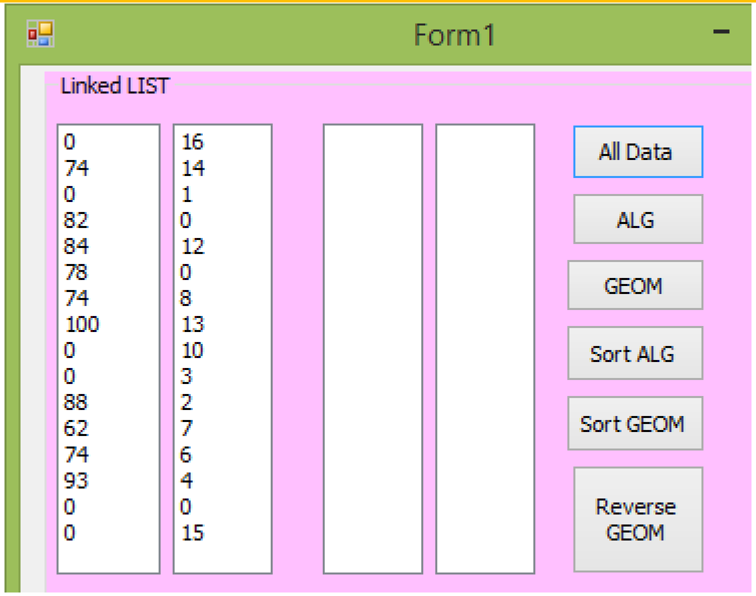
Program using C#



Program using C#

```
public int n=16,ALG=11,GEOM=5,AVAIL=9;
public int[] TEST = {0,0, 74, 0, 82, 84, 78, 74, 100, 0, 0, 88, 62, 74, 93, 0, 0};
public int[] LINK = {0,16,14,1,0,12,0,8,13,10,3,2,7,6,4,0,15};
```

Program using C#



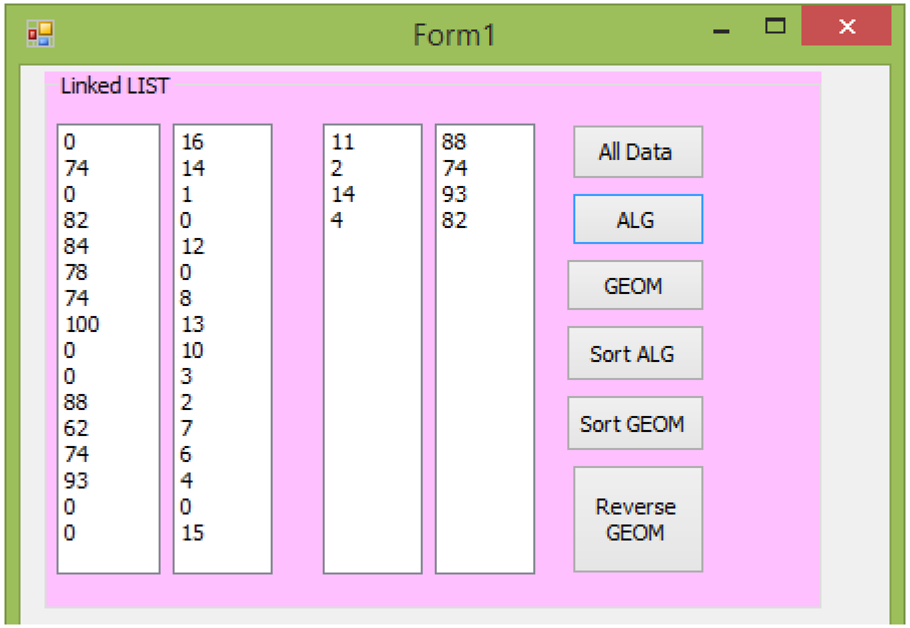
```
0	16
74	14
0	1
82	0
84	12
78	0
74	8
100	13
0	10
0	3
88	2
62	7
74	6
93	4
0	0
0	15
```

```
button_Click()
{
    listBox1.Items.Clear();
    listBox2.Items.Clear();
        for (int i = 1; i <=n; i++)
          {
          listBox1.Items.Add(TEST[i]);
          listBox2.Items.Add(LINK[i]);
          }
}
```

Program using C# Print LIST

```
public void print_list(int[] info, int[] link, int start)
        {
            listBox3.Items.Clear();
            listBox4.Items.Clear();
            int ptr = start;
            while (ptr != 0)
            {
                listBox3.Items.Add(ptr);
                listBox4.Items.Add(info[ptr]);
                ptr = link[ptr];
            }
        }
```

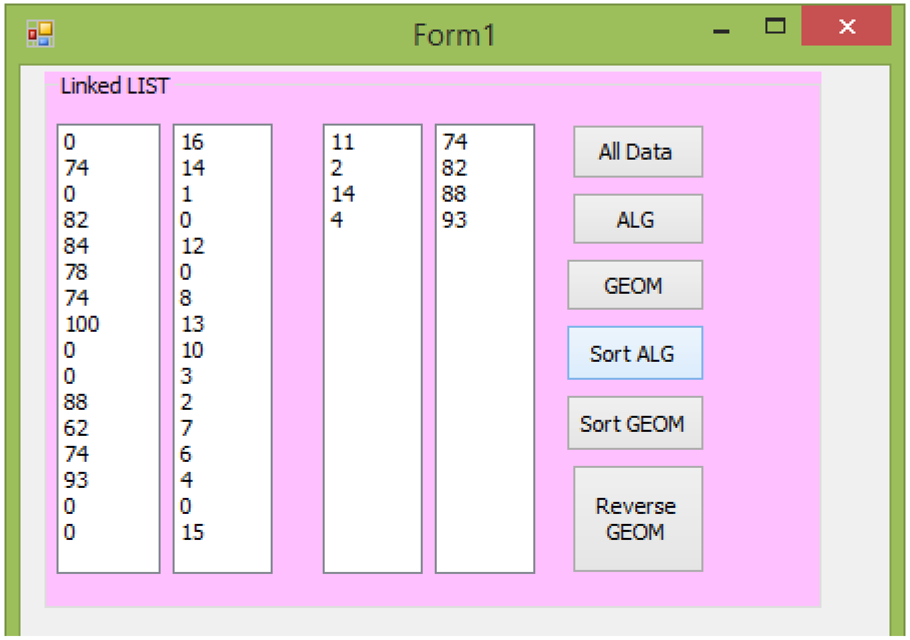
Program using C#



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

```
private void button9_Click(object sender, EventArgs e)
        {
            print_list(TEST, LINK, ALG);
        }
```

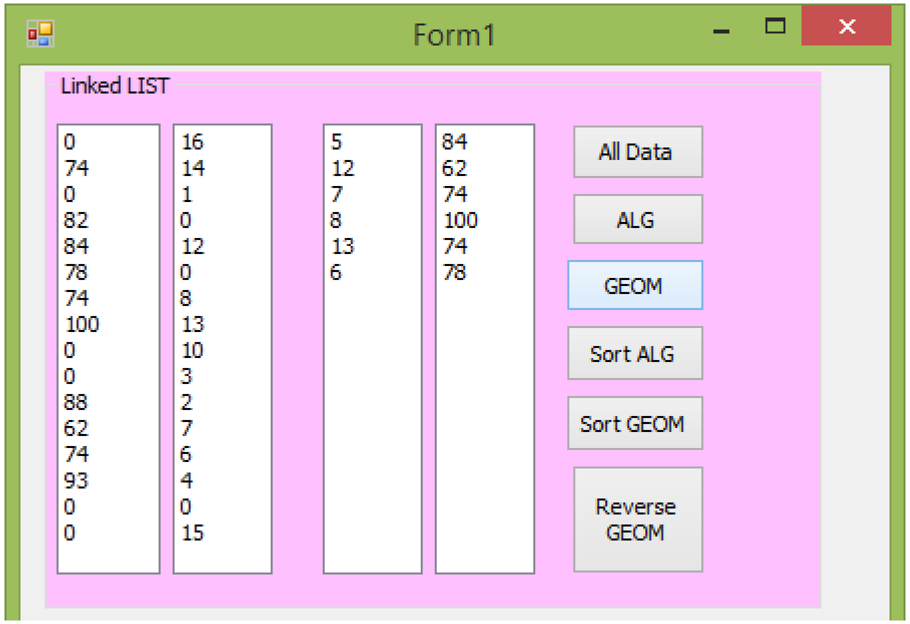
Program using C#



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

```
private void button11_Click(object sender, EventArgs e)
        {
            selection_sort(TEST, LINK, ALG);
            print_list(TEST, LINK, ALG);
        }
```

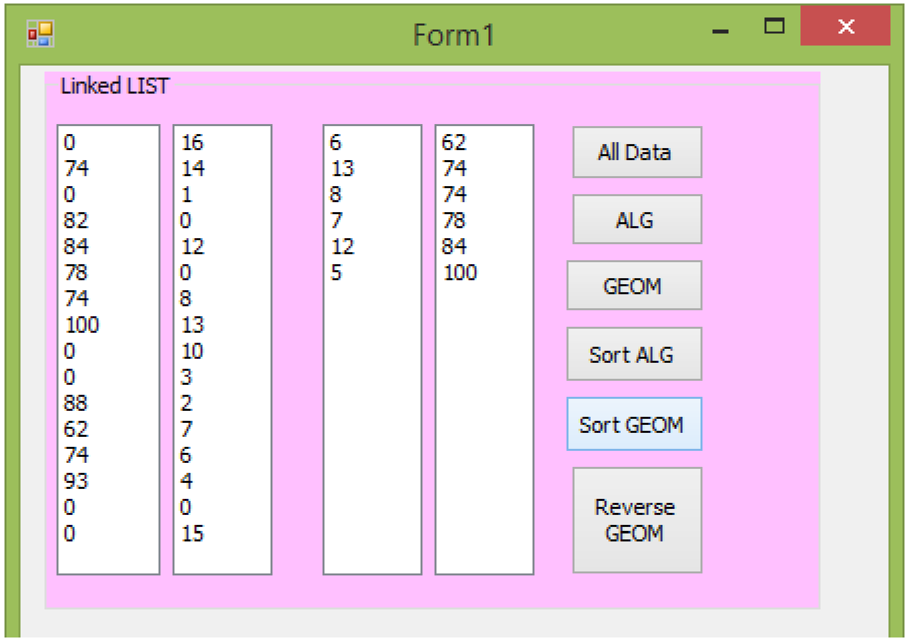
Program using C#



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

```
private void button10_Click(object sender, EventArgs e)
        {
            print_list(TEST, LINK, GEOM);
        }
```

Program using C#

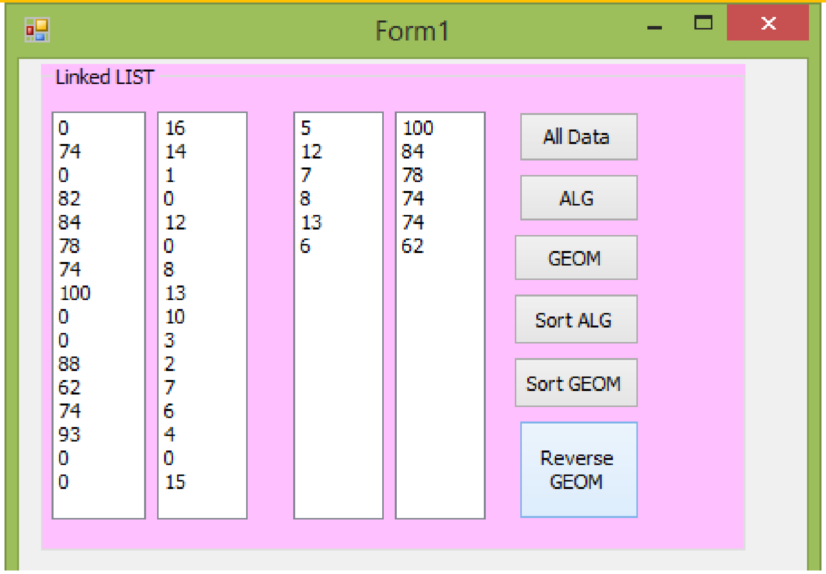

Form1

Linked LIST

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

```
private void button12_Click(object sender, EventArgs e)
        {
            selection_sort(TEST, LINK, GEOM);
            print_list(TEST, LINK, GEOM);
        }
```

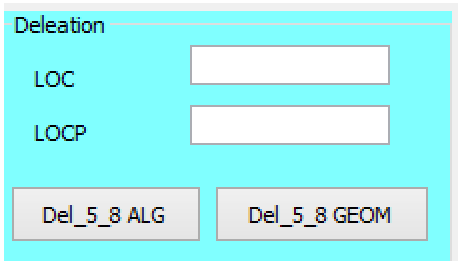
Program using C#



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

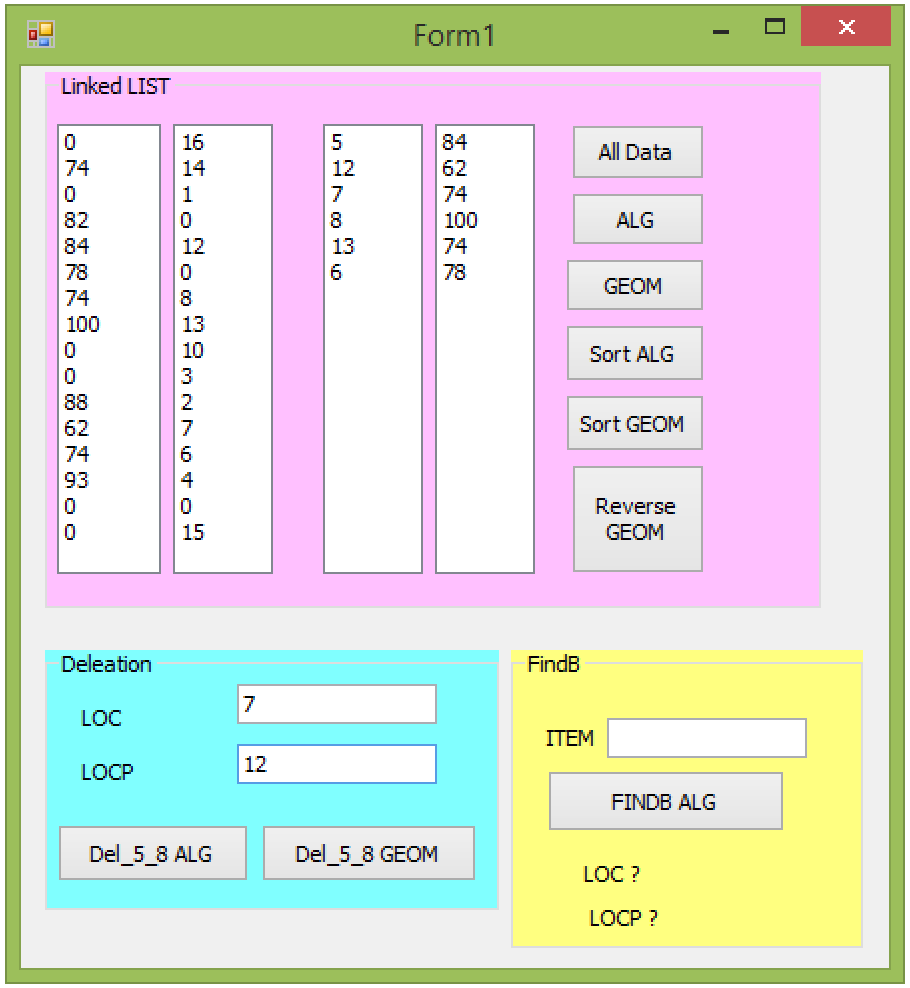
```
private void button13_Click(object sender, EventArgs e)
       {
            GEOM=reverse(TEST, LINK, GEOM);
            print_list(TEST, LINK, GEOM);
       }
```

Program using C# Del



```
private void button15_Click(object sender, EventArgs e)
        {
            int LOC;
            int LOCP;
            LOC = int.Parse(textBox2.Text);
            LOCP = int.Parse(textBox3.Text);
            del(TEST, LINK, ALG, AVAIL, LOC, LOCP);
            print_list(TEST, LINK, ALG);
        }
```

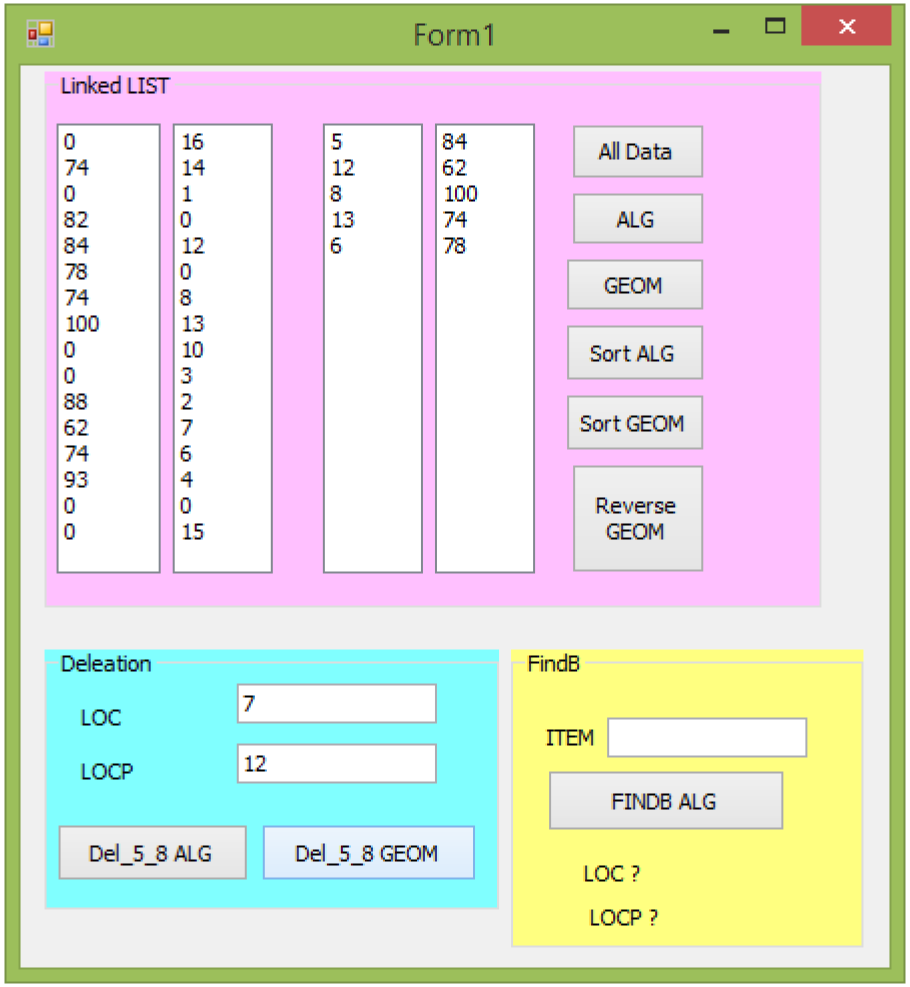
Program using C# Del



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

LOC: 7

LOCP: 12

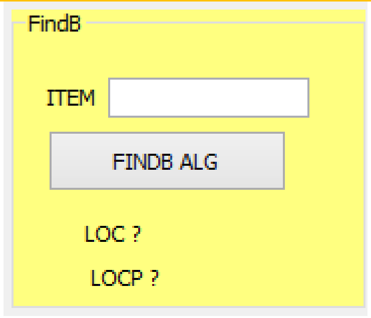


```
0     16    5     84
74    14    12    62
0     1     8     100
82    0     13    74
84    12    6     78
78    0
74    8
100   13
0     10
0     3
88    2
62    7
74    6
93    4
0     0
0     15
```

LOC: 7

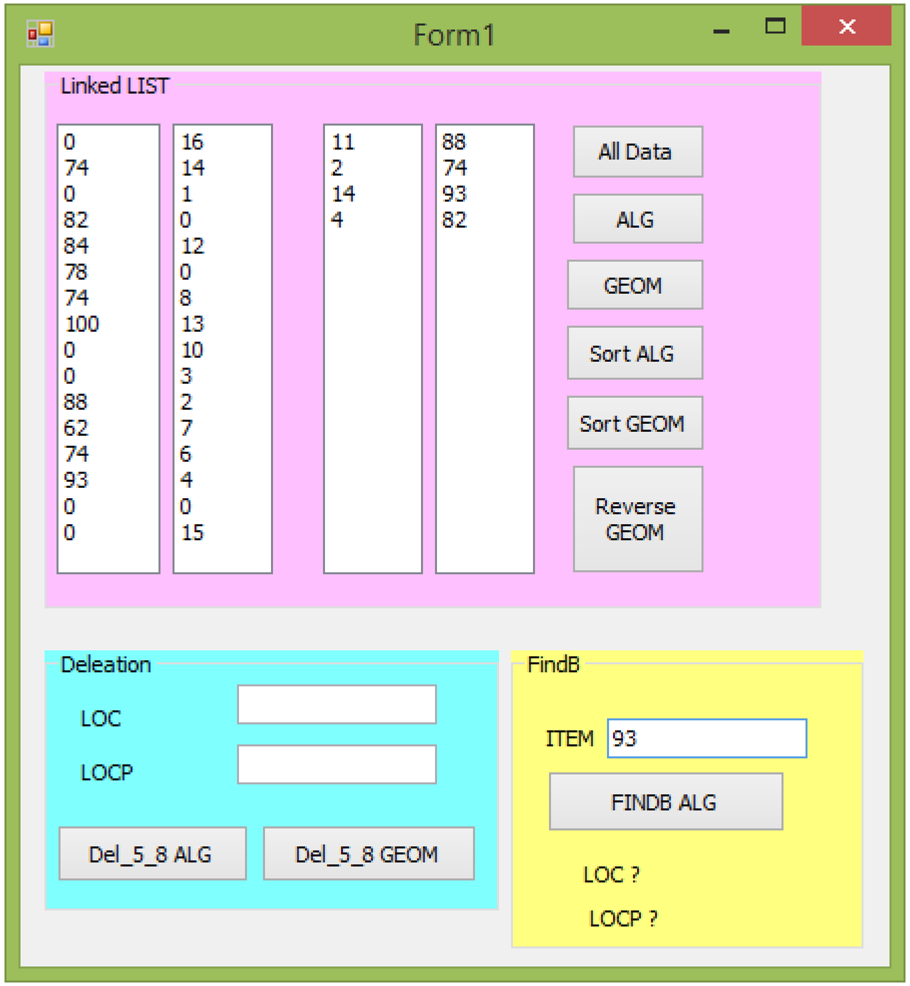
LOCP: 12

Program using C# Find and Del



```
private void button16_Click(object sender, EventArgs e)
        {
            int LOC, LOCP,ITEM;
            int[] Location = new int[2];
            ITEM = int.Parse(textBox7.Text);
            Location = findb(TEST, LINK, ALG, ITEM);
            LOC = Location[0];
            LOCP = Location[1];
            label19.Text = "LOC=" + LOC;
            label20.Text = "LOCP=" + LOCP;
            del(TEST, LINK, ALG, AVAIL, LOC, LOCP);
            print_list(TEST, LINK, ALG);
        }
```

Program using C#



Form1

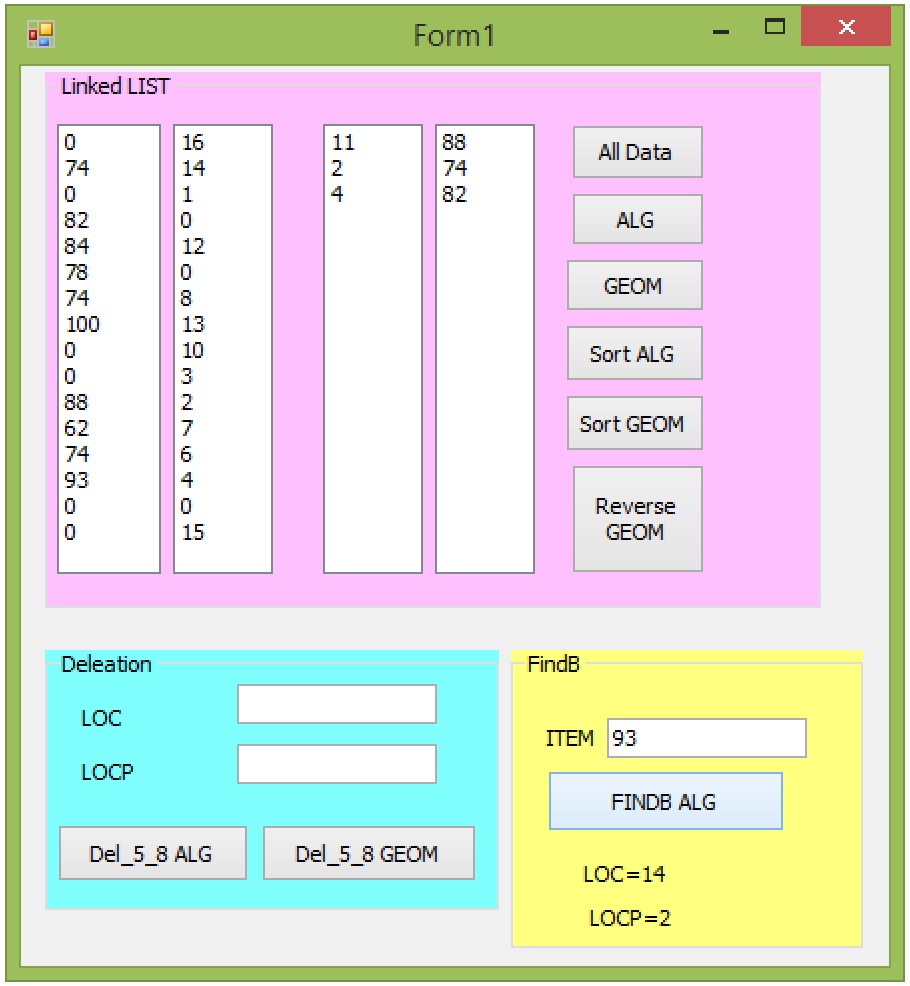
Linked LIST

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

ITEM 93

LOC ?

LOCP ?



Form1

Linked LIST

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

ITEM 93

LOC=14

LOCP=2

تم الإنتهاء من المحاضرة