Data Structures and Algorithm Analysis

3

Dr. Syed Asim Jalal
Department of Computer Science
University of Peshawar

Linked List

Introduction to Linked Lists

Limitations of Arrays as Lists

- We have seen implementation of a list by Array. It is a property of an array that its elements are placed consecutively in memory.
- There may not be enough continuous memory locations to accommodate entire Array. Even if there are more free memory locations scattered throughout memory in the form of small free blocks.
- Once we declare size of Array, it is not possible to increase or decrease it during the execution of the program. If we need more elements to store in the Array, there is need of changing its size in the declaration.

- Usually the number of items to be placed in any list is not known in advance in most cases.

- If Array is used to implement a List, it could result in wastage of memory, if the entire array space is not consumed.

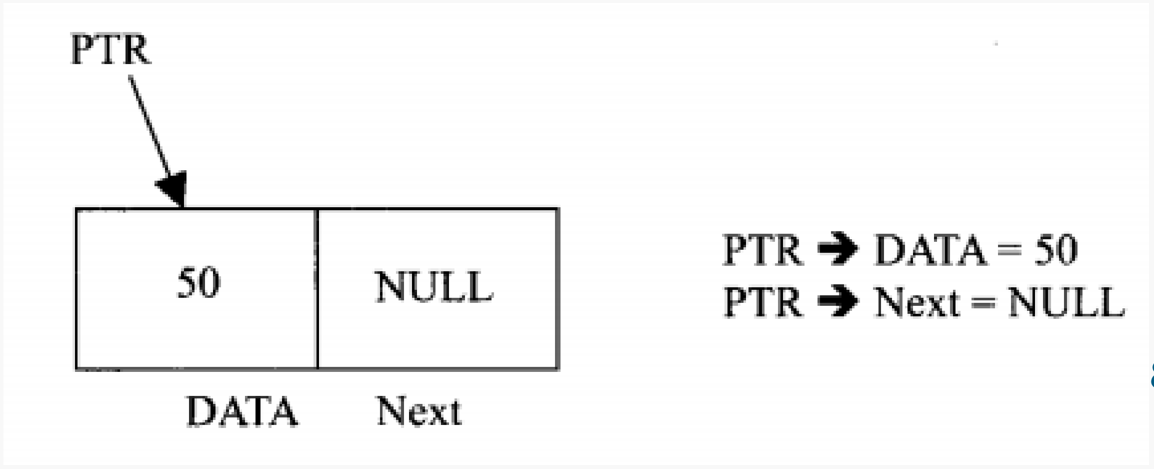
- In order to overcome these shortcomings of a Arrays as a List we need an implementation of a List that
    - would not require static list size and
    - the memory is not reserved from the start. That is, the memory is dynamically assigned as the need arises.
    - Furthermore, it would not require continuous large free memory block to store list and will be able to use scattered smaller free memory blocks.
- Linked List implementation of a List is one such example.

Linked List

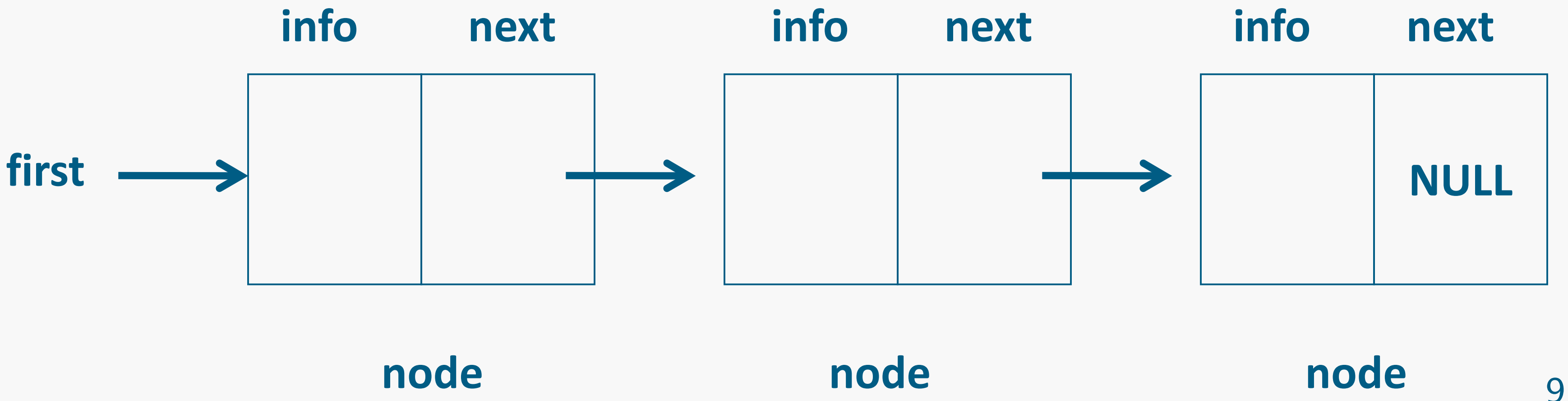
- Linked List is an implementation of a list using linked memory.
- Linked List overcomes the limitations of Array based list implementation.
- A linked list is a dynamic data structure that grows and shrinks without any limitation on size (except total memory space)

- Linked List is a linear collection of data elements.
- Each data element in a Linked List is called a Node.
- The Linear order is not implemented by continuous memory locations but through pointers linking data elements. The next element to a node may be stored anywhere in the memory. Linked List keep track of next element’s location.

- To create a linked list, at first, we define structure of individual elements or the node.
- Each node is divided into two parts.
  - Information Part
  - Link Field or Next Pointer
- The Information part can again consist of many data items.
- The Next or Link pointer field holds the starting location of the next node.

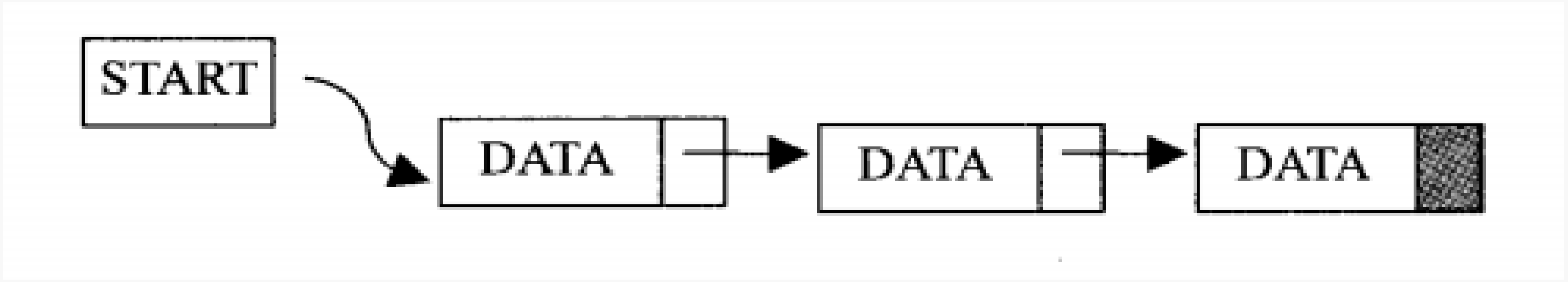


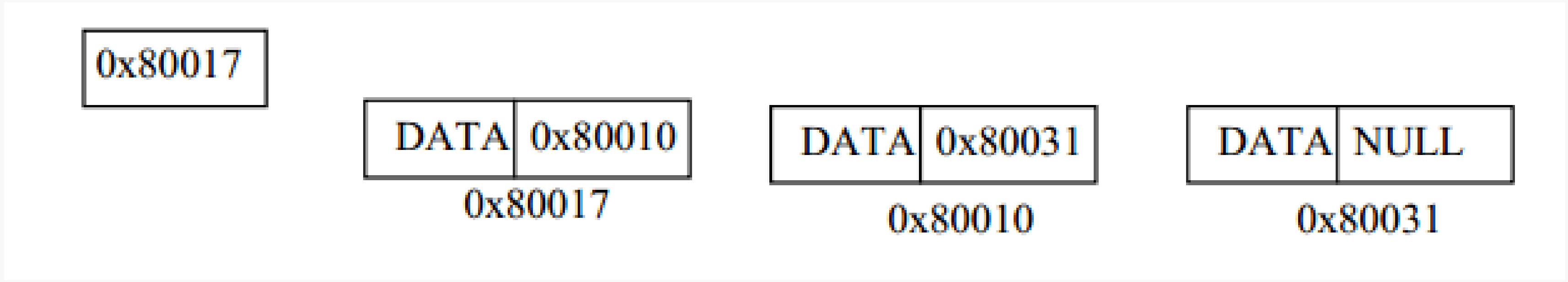
- Different nodes may occur at different locations in main memory, depending on where the operating system assigned memory, but the **Next** part of each node always contain the address of the next node. Thus it forms a chain of nodes which we call a Linked List.



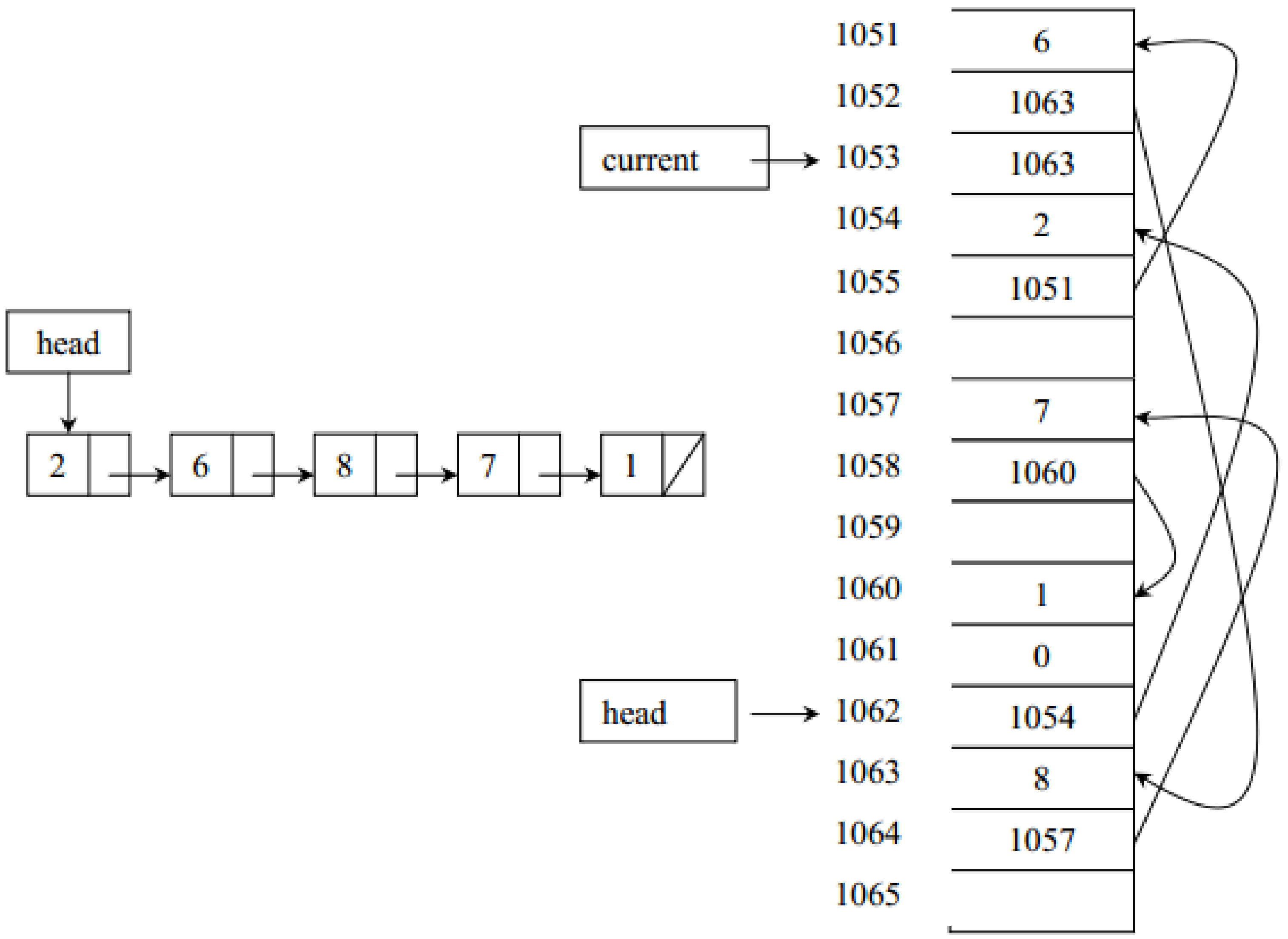
- The **Next** pointer in Last Node
  - If there is no next node, that is, the node is the last node, then the next part has Null Pointer, which points to nothing.

- The **Start** pointer
  - In Linked List we always have a START pointer that always points to First Node.
  - We also call it Head Pointer
  - It has address of the first node of a Linked List.
  - Without Start or Head pointer, it will not be possible to know the starting position of a list.

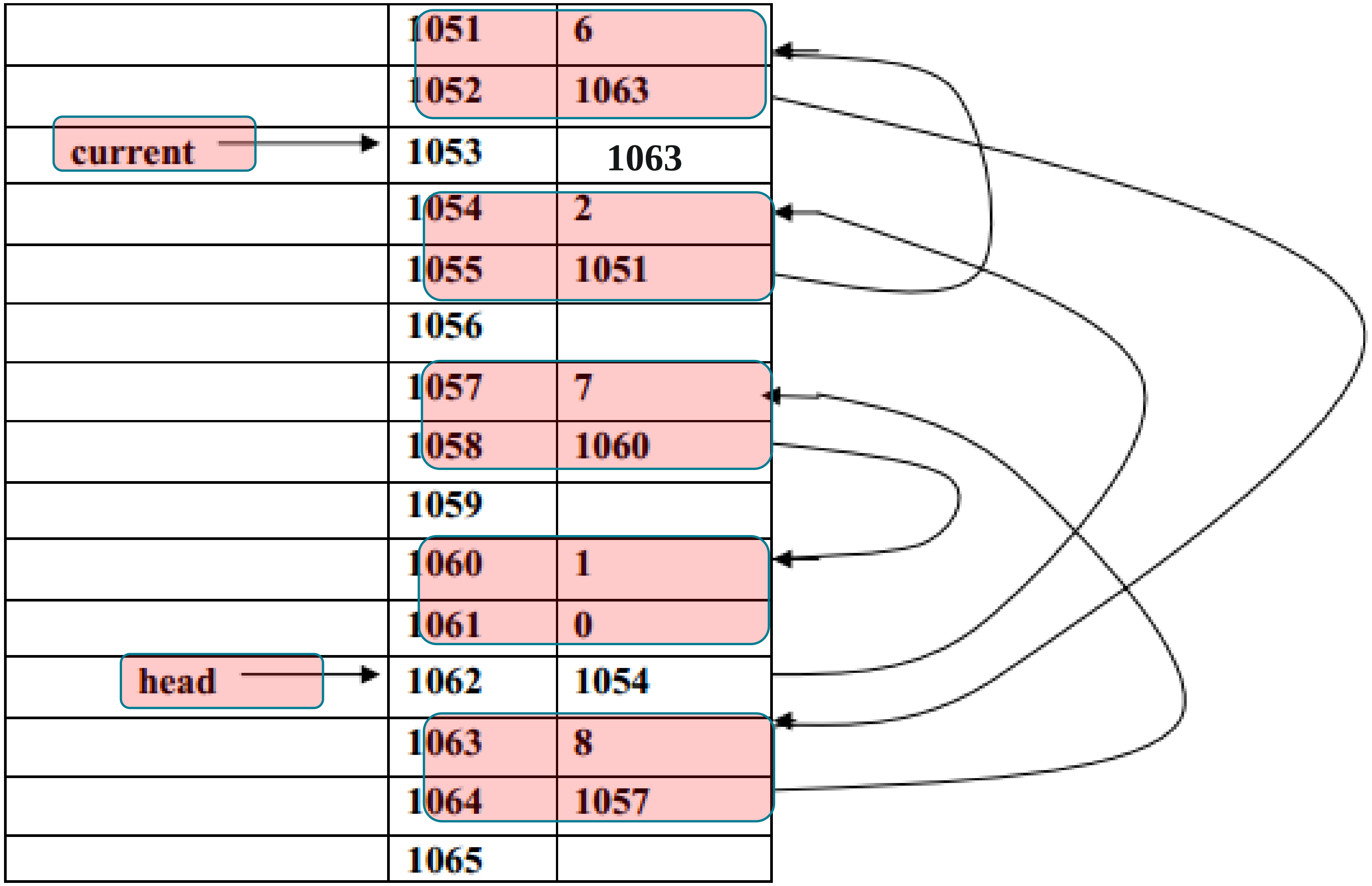

START
DATA
DATA
DATA


0x80017
DATA
0x80010
0x80017
DATA
0x80031
0x80010
DATA
NULL
0x80031

Linked List inside Computer Memory



Linked List in memory: scattered nodes



	1051	6
	1052	1063
current	1053	1063
	1054	2
	1055	1051
	1056	
	1057	7
	1058	1060
	1059	
	1060	1
	1061	0
head	1062	1054
	1063	8
	1064	1057
	1065	

Advantages of Linked List

1. Linked list are dynamic data structures. That is, they can grow or shrink during the execution of a program.
2. Efficient memory utilization: In linked list (or dynamic) representation, memory is not pre-allocated. Memory is allocated whenever it is required. And it is de-allocated (or removed) when it is not needed.
3. Insertion and deletion in middle of a List are efficient. Linked list provides flexibility in inserting a data item at a specified position and deletion of a data item from the given position.
4. Many complex applications can be easily carried out with linked list.

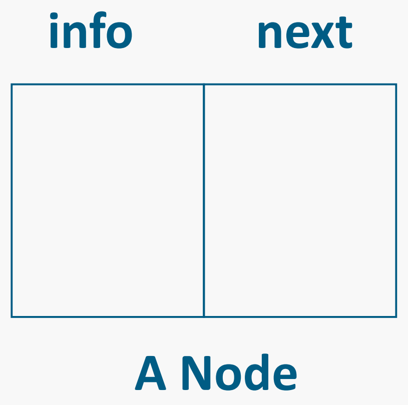
Disadvantages of Linked List

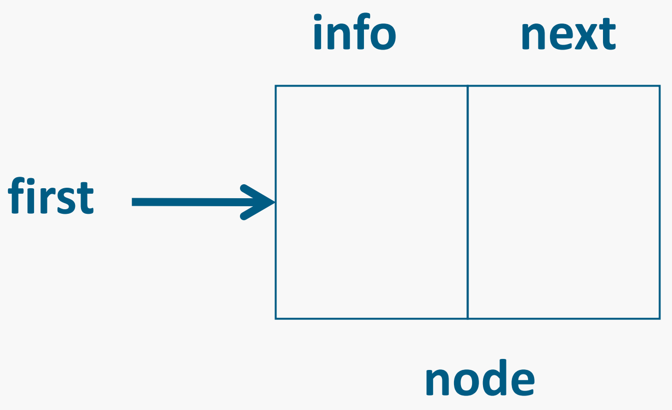
1. Memory overhead.
   - to store an integer number, a node with integer data and address field is allocated. That is more memory space is needed.
   - It is called overhead memory.

2. Access to an arbitrary data item is little bit cumbersome and also time consuming.
   - As readily available indexes like arrays are not available. We have to traverse list to access desired node.

Implementation of Node

```
struct node
{
   int info;
   node * next;
};
struct   node   *first;
```





Operations on Linked List

- Creation of Linked List
- Insertion of an element
- Deletion of an element
- Traversing all elements
- Searching for data

- **Creation operation** is used to create a linked list.
- Usually it means creating the first node.
- Once a linked list is created with one node, insert operation can be used to add more elements in a linked list.
- Here, creating a Node means allocating memory and returning its memory address to be stored Start pointer.

- **Insertion operation** is used to insert a new node at any specified location in a linked list.
- A new node may be inserted.
  - (a) At the beginning of the linked list
  - (b) At the end of the linked list
  - (c) At any other specified position

- **Deletion operation** is used to delete an item (or node) from a linked list.
- A node may be deleted from the
  - (a) Beginning of a linked list
  - (b) End of a linked list
  - (c) Specified location of the linked list

- **Traversing** is the process of going through all the nodes from one end to another end of a linked list.
- In a singly linked list we can visit from left to right only, or forward traversing. In a doubly linked list forward and backward traversing both are is possible.
- Singly and doubly linked lists are two types of Linked Lists

TYPES OF LINKED LIST

- 1. Singly linked list
- 2. Doubly linked list
- 3. Circular linked list

Operations on Singly Linked List

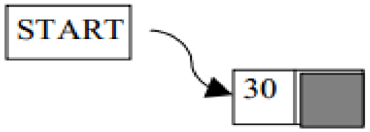


Fig. 5.5. Create a node with DATA(30)

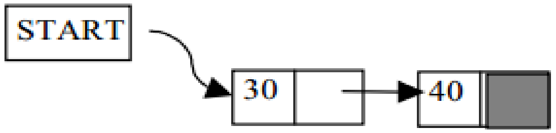


Fig. 5.6. Insert a node with DATA(40) at the end

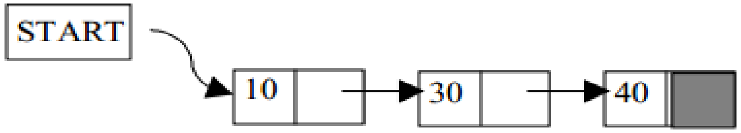


Fig. 5.7. Insert a node with DATA(10) at the beginning

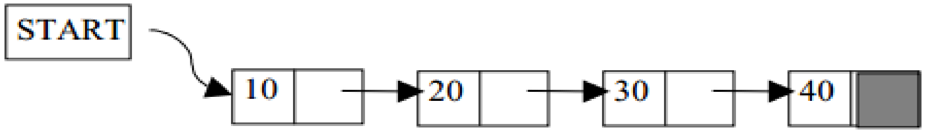


Fig. 5.8. Insert a node with DATA(20) at the 2nd position

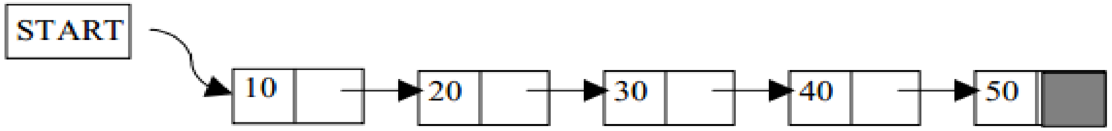


Fig. 5.9. Insert a node with DATA(50) at the end

Output → 10, 20, 30, 40, 50

Fig. 5.10. Traversing the nodes from left to right

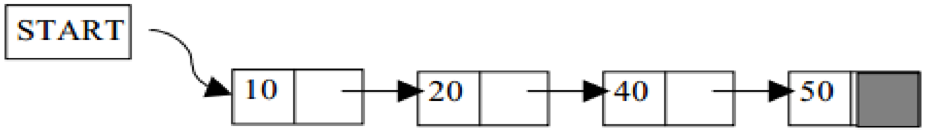
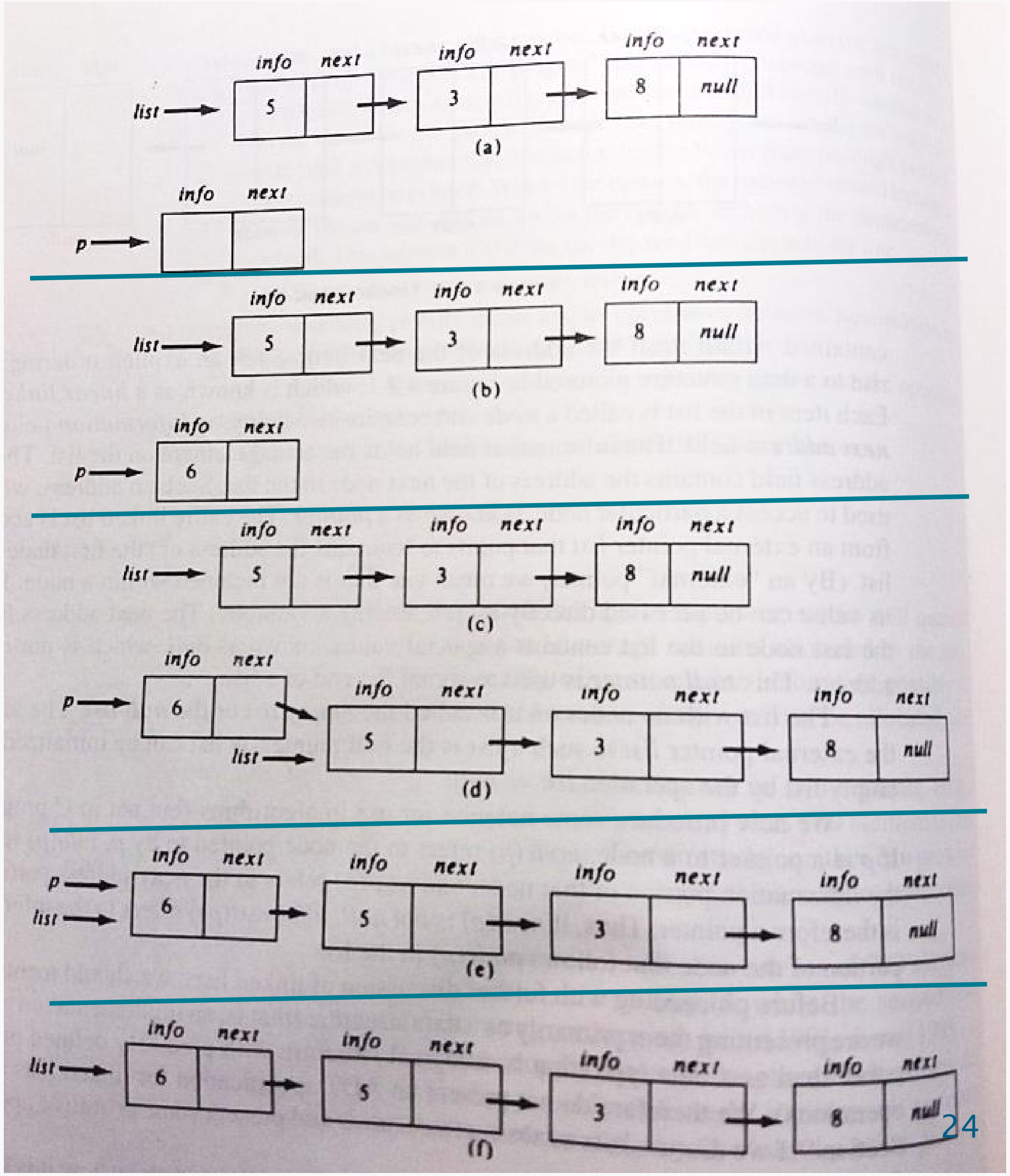


Fig. 5.11. Delete the 3rd node from the list

Graphical representation of FIVE steps involved in Inserting node at the front of a linked list.

Figures:
a, b, c, d, e, and f



Insert a Node at the Beginning & Create Linked List

- Suppose **START** pointer contains address of first node in the Linked List. **INFO** is the information part of a Node. And **DATA** be the information being entered.

```
1. Input DATA to be inserted
2. Create a new linked list node and save address in
"NewNode" pointer
3. NewNode → INFO = DATA
4. If (START equals to NULL)
        (a) Set NewNode→Link = NULL
5. Else
        (a) Set NewNode →Link = START
6. START = NewNode
7. Exit
```

Insert a Node at the end of Linked List

Suppose START points to the first node in a linked list.
INFO is the information part of a Node. And DATA be the information being entered.

```
1. Input DATA to be inserted
2. Create a new node, save address in NewNode
3. Set NewNode →INFO= DATA
4. Set NewNode →Next = NULL

5. If START contains NULL      // linked list is empty
        (a)START = NewNode
6. else
        (a)TEMP_Ptr = START
        (b) While (TEMP_Ptr→Next not equal to NULL)
                (i) TEMP_Ptr = TEMP_Ptr →Next
        (c) TEMP_Ptr→Next = NewNode
8. Exit
```