Linked List

Unit: 5
Hours: 6
Marks: 8

Linked List

- Concept and Definition
- Inserting and Deleting nodes

✓Linked implementation of a stack (PUSH/POP)

- Linked implementation of a queue (Insert/Remove)
- Circular List
  - Stack as a circular list (PUSH/POP)
  - Queue as a circular list (Insert/Remove)
- Doubly Linked List (Insert/Remove)

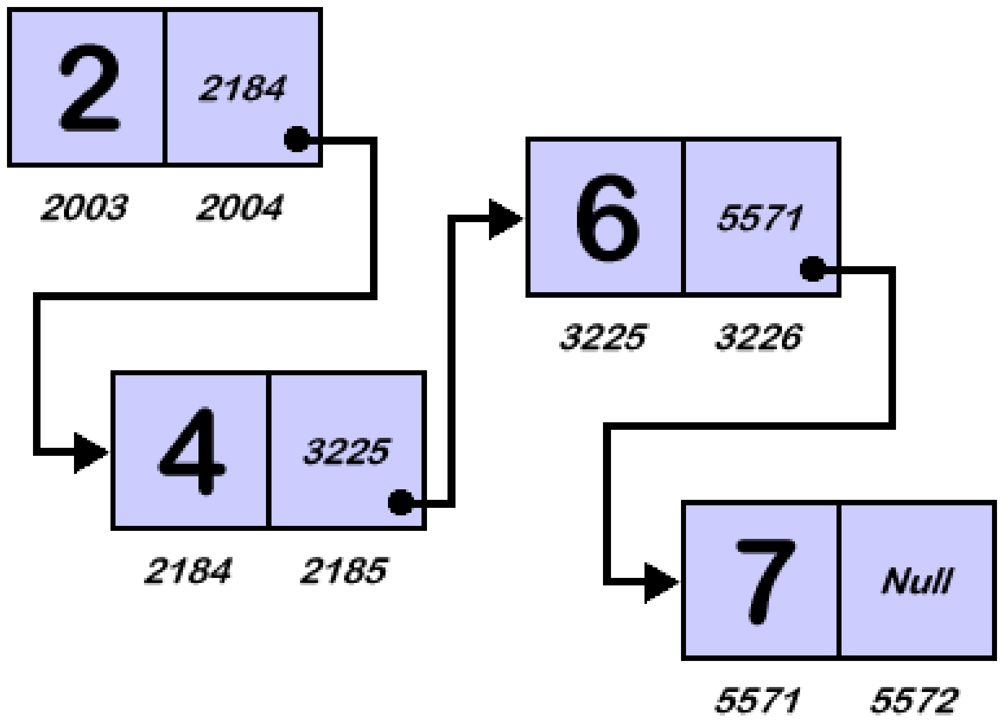
Concept and Definition

- If the memory is allocated before the execution of a program, it is fixed and cannot be changed.
- Linked list provides flexibility on storage system.
- No Use of Array.

- When we allocate memory using array, memory may not be fully utilized.
- Linked list provides mechanism to allocate memory whenever required.

Linked List

- Linked list are special list of some elements linked to one another.
- Each element point to other element.
- Each element is called NODE.
- Each node contains two parts
  - INFO: Stores information
  - POINTER: points to next node



KEY TERMS

- DATA Field: Stores information
- LINK Field: Stores pointer
- NULL pointer: Last node contains null which indicates the end of list.
- External Pointer: pointer to the very first node of the list. Enables us to access entire linked list.
- Empty List: Nodes are not present in the list.

Advantages

- **Linked list are dynamic data structures**. That is they can grow or shrink during the execution of a program.
- **Efficient Memory Utilization:** Memory is allocated when ever required.
- **Insertion and deletion are easier and efficient:**
- Many complex applications can be easily carried out with linked lists.

Disadvantages

- More memory: If the number of fields are more, then more memory space is needed.
- Access to an arbitrary data item is little bit cumbersome and also time consuming.

Representation of Linear Linked List

```
Struct node
{
        int a;
        Struct node *next;
};
Typedef struct node NODE;
NODE *start;
```

Operations on Linked List

- Creation
- Insertion
- Deletion
- Traversing
- Searching
- Concatenation
- Display

Operations: Creation

- Used to create a linked list

Operations: Insertion

- used to insert a new node in the linked list at the specified position.
- A new node may be inserted
  - At the beginning of the linked list.
  - At the end of a linked list
  - At the specified position in a linked list
  - If the list itself is empty, then the new node is inserted as a first node.

Operations: Deletion

- This operation is used to delete an item(a node) from the linked list.
- A node may be deleted from the
  - Beginning of a linked list
  - End of a linked list
  - Specified position in the list.

Operation: Traversing

- It is the process of going through all the nodes of a linked list from one end to the other end.
- If we start traversing from the very first node towards the last node it is called forward traversing.
- If the desired node is found we signal operation “successful” otherwise we signal it is unsuccessful.

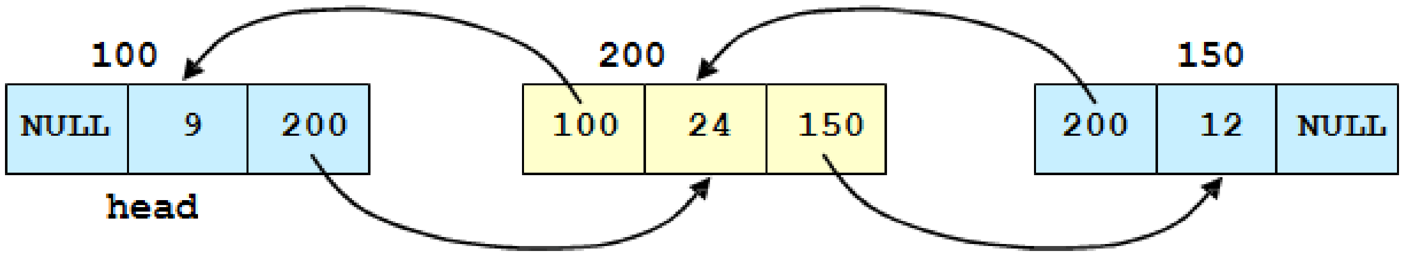
Concatenation

- It is the process of appending (Joining) the second list to the end of the first list consisting of m nodes.
- When we concatenate two lists, if the second list has n nodes, Then concatenating list will be having m+n nodes.
- The last node of the first list will be modified to point to the first node of the second list.

Display:

- This operation is used to print each and every node’s information.
- We access each node from the beginning or the specified position of the list and output the data housed there.

Types of Linked List

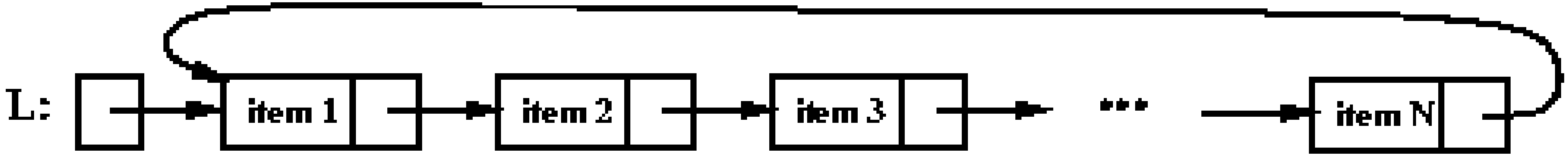


Şekil 2.8 Silinmek istenen ortadaki düğüm sarı renkte gösterilmiştir.

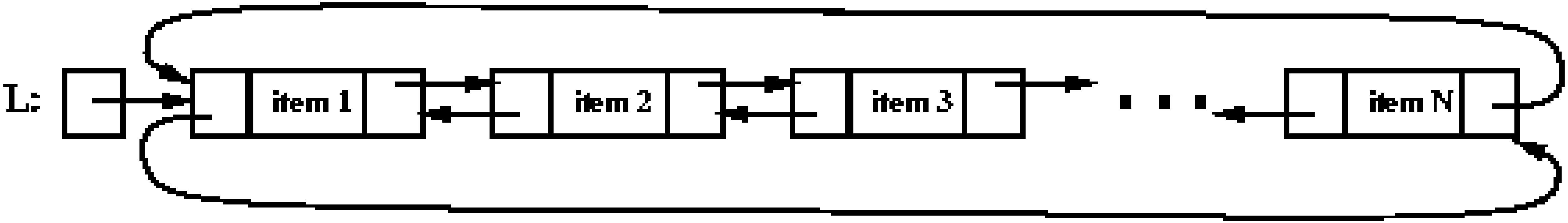
- Singly Linked List
  - Node have one pointer of next Node
- Doubly Linked List
  - Node have two pointers of next Node and Previous Node.
- Circular Linked List
  - No beginning no end. Last node points to the very first node.
- Circular Doubly Linked List:
  - Last node points to the very first node in the next field and first node points to last node as previous node.

Types of Linked List

Circular, singly linked list:



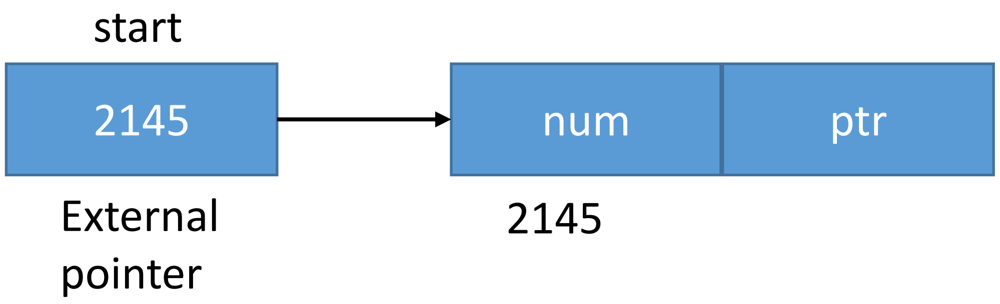
Circular, doubly linked list:



Singly Linked List

- every element contains some **data** and a **link** to the next element
- All nodes are linked together in sequential manner
- It has beginning and the end
- Problem:
  cannot access predecessor of node from the current node

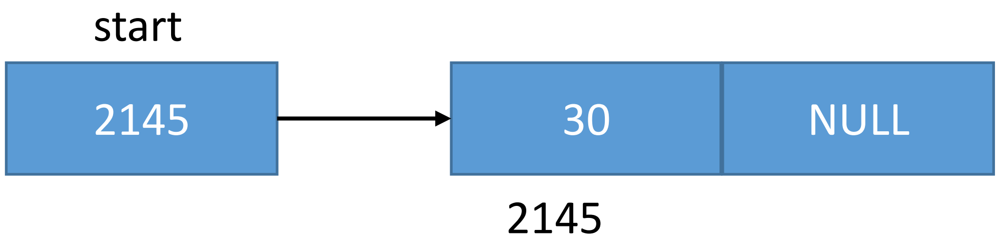
Singly Linked List




```
Struct node
{
        int num;
        Struct node *ptr;                       /*pointer to next node*/
};
Typedef struct node NODE;                       /*type definition making it abstract data type*/
NODE *start;                                    /*pointer to the node of linked list*/
Start = (NODE *) malloc (sizeof (NODE));        /*dynamic memory allocation */
```

<u>**Now we can assign values**</u>

```
Start -> num = 30;
Start -> ptr = NULL;
```



Inserting Nodes

- Allocating a node
- Assigning the data
- Adjusting the pointers

- 1. Insertion at the beginning of the list
- 2. Insertion at the end of the list
- 3. Insertion at the specified position within the list.

Steps for Inserting an element to linked list

Step-1: Get the value for NEW node to be added to the list and its position.

Step-2: Create a NEW, empty node by calling malloc(). If malloc() returns no error then go to step-3 or else say "Memory shortage".

Step-3: insert the data value inside the NEW node's data field.

Step-4: Add this NEW node at the desired position (pointed by the "location") in the LIST.

Step-5: Go to step-1 till you have more values to be added to the LIST.

An algorithm to insert a node at the beginning of the singly linked list:

let *head be the pointer to first node in the current list

1. Create a new node using malloc function

```
NewNode = (NodeType*) malloc (sizeof(NodeType));
```

2. Assign data to the info field of new node

```
NewNode->info = newItem;
```

3. Set next of new node to head

```
NewNode->next = head;
```

4. Set the head pointer to the new node

```
head = NewNode;
```

5. End

An algorithm to insert a node at the end of the singly linked list:

let *head be the pointer to first node in the current list

1. Create a new node using malloc function

   NewNode=(NodeType*)malloc(sizeof(NodeType));

2. Assign data to the info field of new node

   NewNode->info=newItem;

3. Set next of new node to NULL

   NewNode->next=NULL;

4. if (head ==NULL)then

   Set head =NewNode and exit.

   Set temp=head;

5 Else set temp = head

   while(temp->next!=NULL)

   temp=temp->next;   //increment temp

7. Set temp->next=NewNode;

8. End

An algorithm to delete the first node of the singly linked list:

let *head be the pointer to first node in the current list

1. If(head==NULL) then
   print “Void deletion” and exit
2. Store the address of first node in a temporary variable
   temp = head;
3. Set head to next of head.
   head=head->next;
4. Free the memory reserved by temp variable.
   free(temp);
5. End

An algorithm to delete the last node of the singly linked list:

let *head be the pointer to first node in the current list

```
1. If(head==NULL) then                          // list is empty
        print "Void deletion" and exit
2. else if(head->next==NULL) then               // list has only one node
        printf("%d" ,head->info);               //print deleted item
        free(head);
3. else
        set temp=head;
        while (temp->next->next != NULL)
                set temp = temp->next;
        free(temp->next);
        Set temp->next=NULL;
4. End
```