# **Module Code:** CSE401

# **Session 12:** Linked List

## Session Speaker:

**Prema Monish**
premamonish.tlll@msruas.ac.in


©Ramaiah University of Applied Sciences

# Session Objectives

- At the end of this lecture, student will be able to
  - use the structure and operations of a ***singly linked list*** data structure
  - use the structure and operations of a ***doubly linked list*** data structure




©Ramaiah University of Applied Sciences

# Session Topic

- Linked Lists implementation
- Double linked list implementation




©Ramaiah University of Applied Sciences

# Disadvantage of arrays

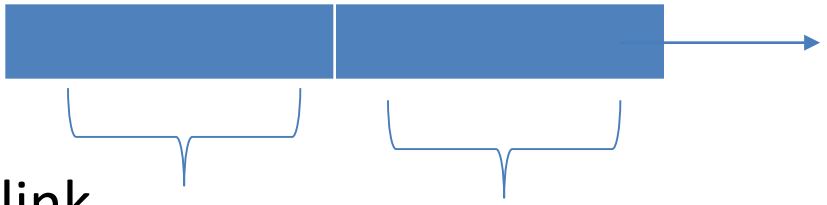
- Array items are stored contiguously.
- Insertion and deletion operations involving arrays is tedious job.
- What happens if the input data is changing dynamically?
- What happens if you want the structure to be always sorted? Is there an efficient way to do it?



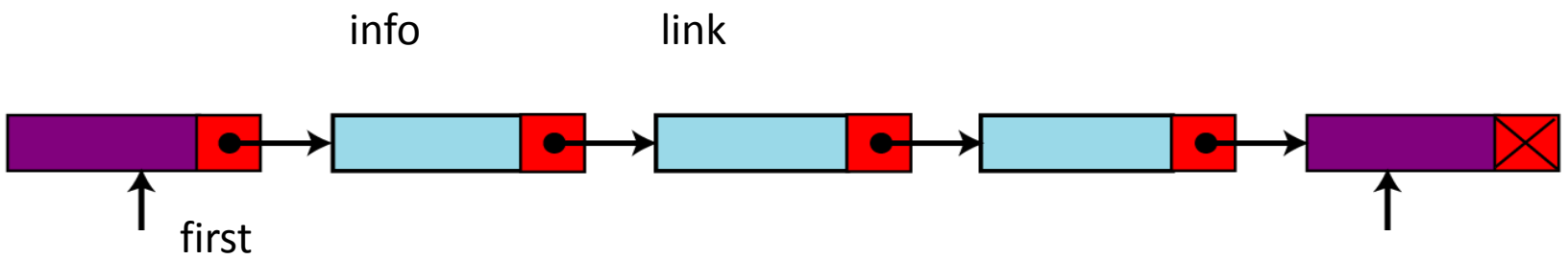

©Ramaiah University of Applied Sciences

# Linked Lists

- Linked List is data structure which is collection of zero or more nodes where each node has some information and address of next node.
- A linked list can grow or shrink in size as the program runs
- The last node points to NULL



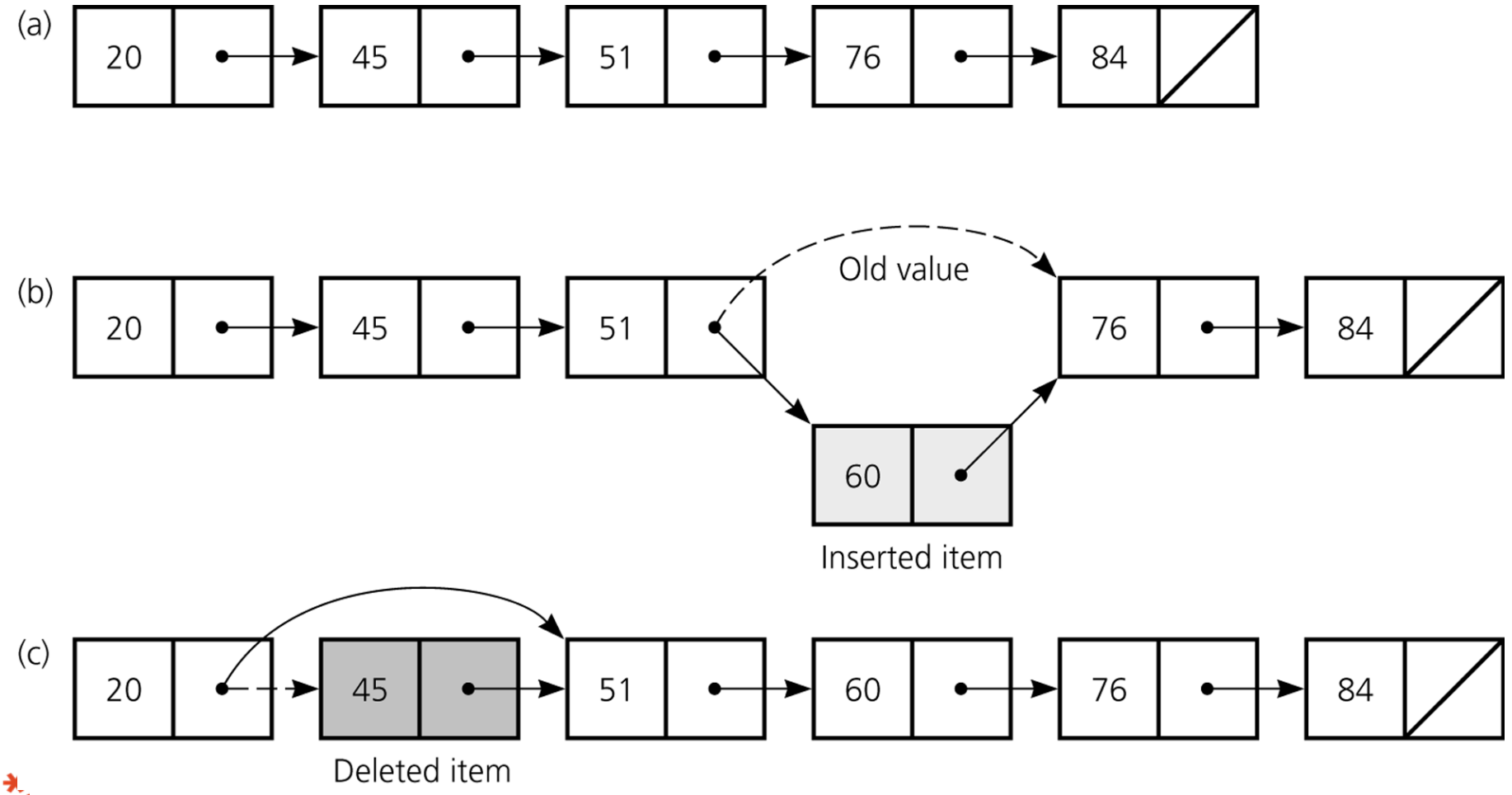

- Node=info+link




- Types of linked list-single linked list and double linked list


©Ramaiah University of Applied Sciences

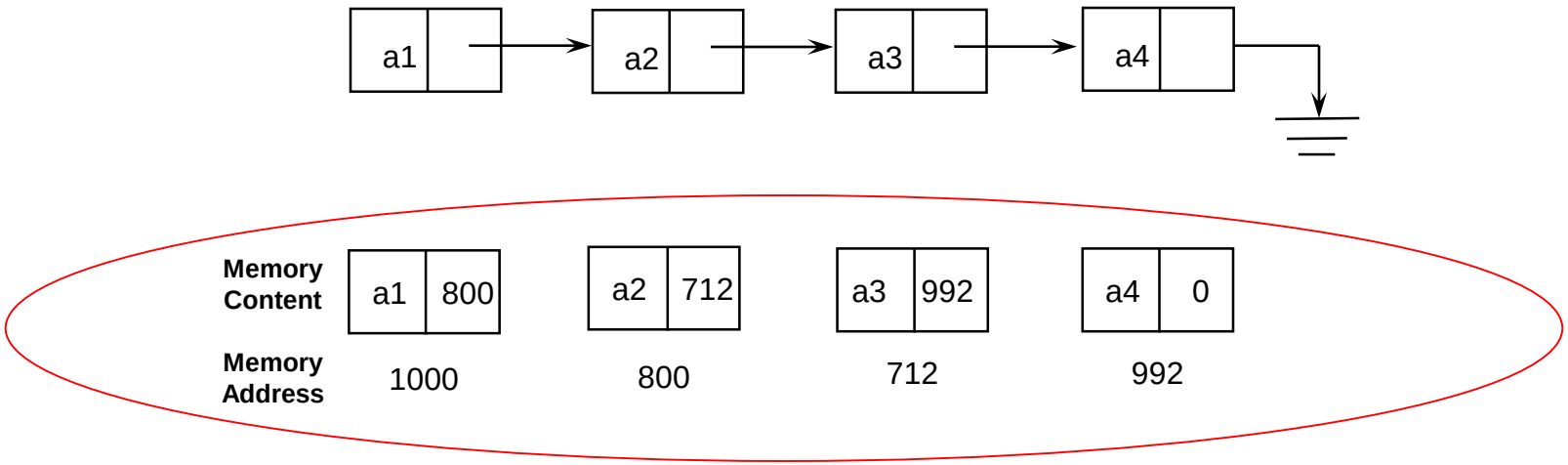
# A Linked List of Integers





©Ramaiah University of Applied Sciences

# What does the memory look like?




- Linked list nodes are normally not stored contiguously in memory
- Logically, however, the nodes of a linked list appear to be contiguous


©Ramaiah University of Applied Sciences

# Array Vs. Linked List

- Lists of data can be stored in arrays, but linked lists provide several advantages

- A linked list is appropriate when
  - the number of data elements to be represented in the data structure is unpredictable
- Linked lists are dynamic
  - length of a list can increase or decrease as necessary
  - The size of an array, cannot be altered once memory is allocated




©Ramaiah University of Applied Sciences

# Array Vs. Linked List contd.

- Linked lists become full only when the system has insufficient memory to satisfy dynamic storage allocation requests
  - Arrays can become full
- An array can be declared to contain more elements than the number of data items expected
  - This can waste memory
  - Linked lists can provide better memory utilization in these situations
- Insertion and deletion in a sorted array can be time consuming
  - In linked list, it is easy




©Ramaiah University of Applied Sciences

# Defining Node

- Since each node has two fields info and link both are different data type.
- Syntax

```
Struct node
{
        Type1 info;
        Type2 *link;
};
```

```
Eg:
Struct node
{
    int info;
    struct node *link;
};
typedef struct node *NODE;//alias name
```




©Ramaiah University of Applied Sciences

# Allocating node

- getnode() is implemented to allocate memory

```
NODE getnode()
{
    NODE x;
    x=(NODE)malloc(sizeof(struct node));
    if(x==null)
    {
        ptintf(“out of memory”);
        return;
    }
    return x;
}
```




©Ramaiah University of Applied Sciences

# De-allocating node

- As malloc is used to  allocate memory it needs to be deallocated

```
void freenode(NODE x)
{
    free(x);
 }
```


©Ramaiah University of Applied Sciences

# Operations on Single linked list

- Inserting a node into the list
- Deleting a node from the list
- Search in a list
- Display the contents of list




©Ramaiah University of Applied Sciences

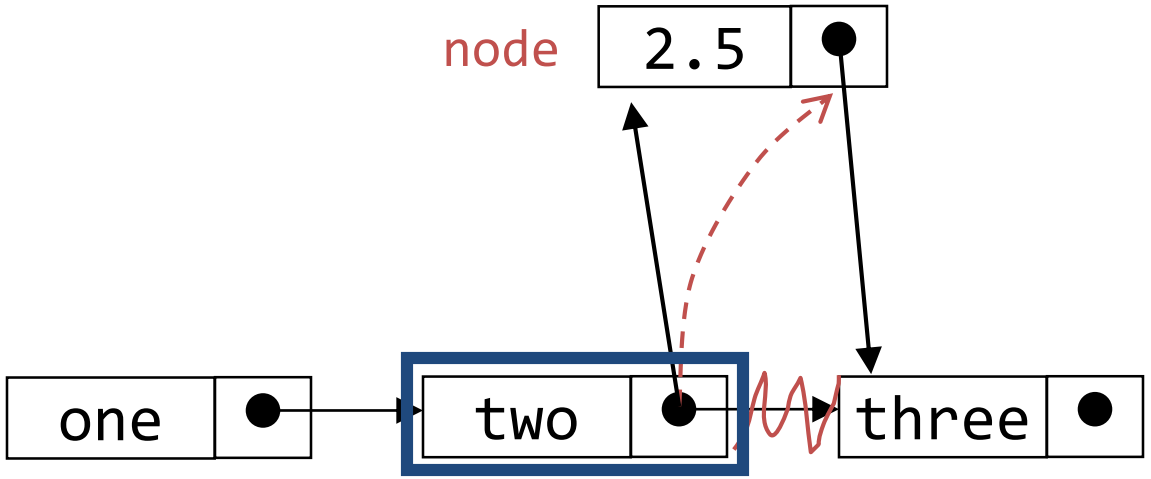
# Inserting Front

```
NODE insert_front(int item,NODE first)
{
    NODE temp;
    temp=getnode();//memory allocated
    temp->info=item;
    temp->link=first;
    return temp;
 }
```




©Ramaiah University of Applied Sciences

# Inserting after



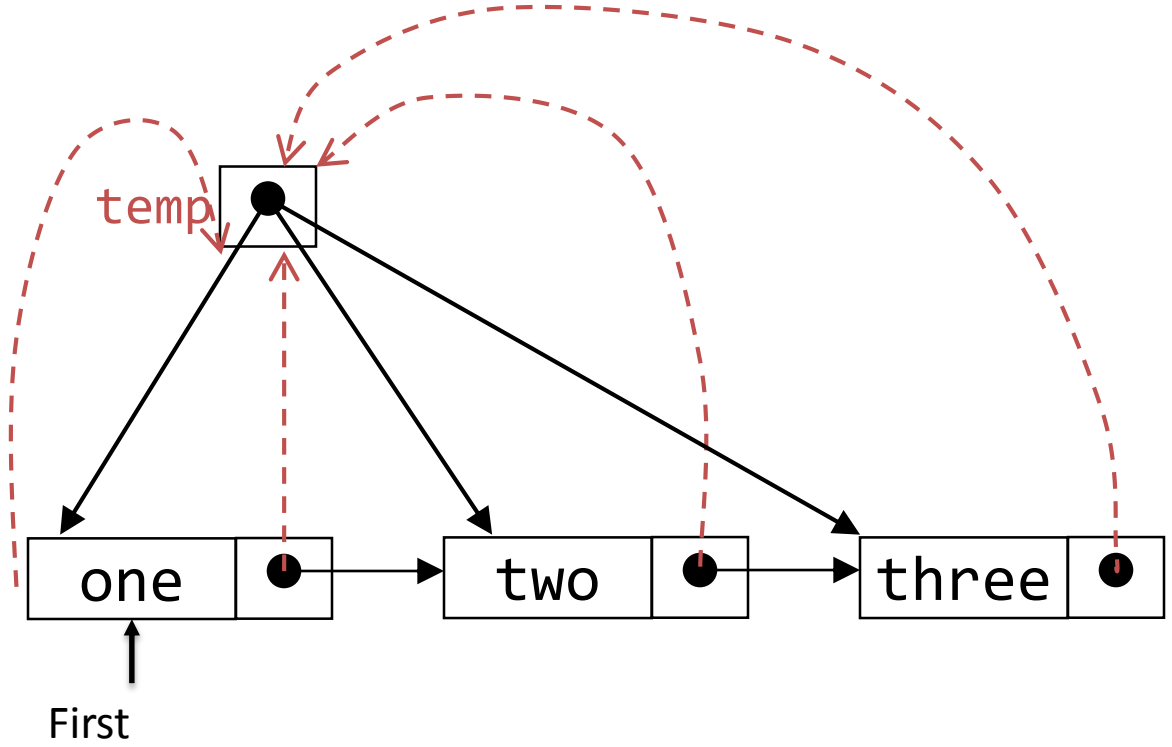

Find the node you want to insert after

***First,*** copy the link from the node that's already in the list

***Then,*** change the link in the node that's already in the list


©Ramaiah University of Applied Sciences

# Display list






©Ramaiah University of Applied Sciences

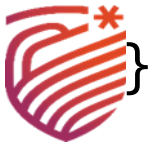
# Display list

```
void display(NODE first)
{
    NODE temp;
    if(first==NULL)
    {
        printf("list empty\n");
        return;
     }
        printf("elements of list are");
        temp=first;
        while(temp!=NULL)
        {
                printf("%d",temp->info);
                temp=temp->link;
        }
}
```


©Ramaiah University of Applied Sciences

# Delete node

```
void delete_front(NODE first)
{
    NODE temp;
    if(first==NULL)
    {
        printf("list empty\n");
        return first;
     }
        printf("elments of list are");
        temp=first;
        temp=temp->link;
        printf("item deleted=%d",first-info);
        freenode(first);
        return temp;
```




©Ramaiah University of Applied Sciences

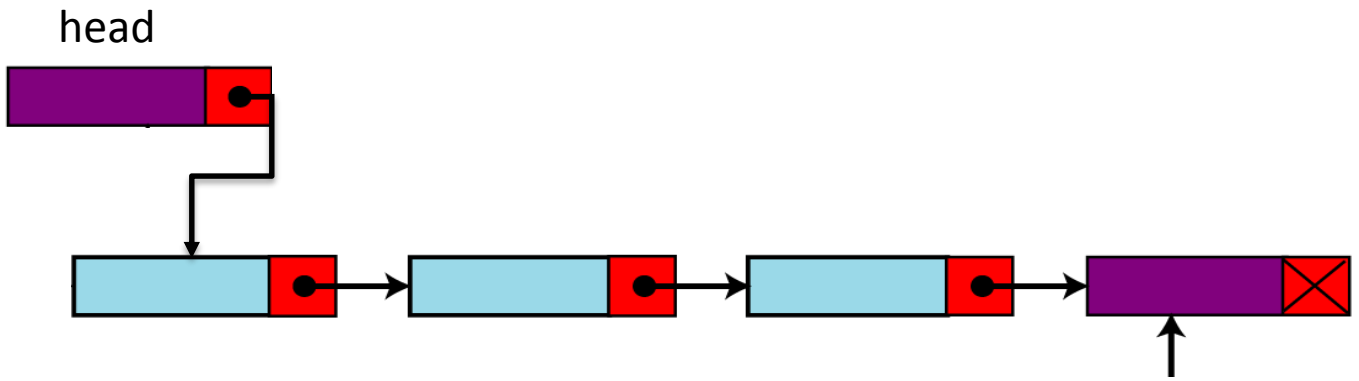
# Searching node

```
void search(int key,NODE first)
{
     NODE cur;
     if(first==NULL)
     {
          printf("list empty\n");
          return ;
      }
          cur=first;
          while(cur!=NULL)
          {
          if(key==cur->info)break;
          cur=cur->link;
          }
          if(cur==NULL)
          {
          printf("search is unsuccessful");
          return;
          }
          printf("search is successful");
}
```


©Ramaiah University of Applied Sciences

# Linked List with Head node

- Head node just points to the first element of list .

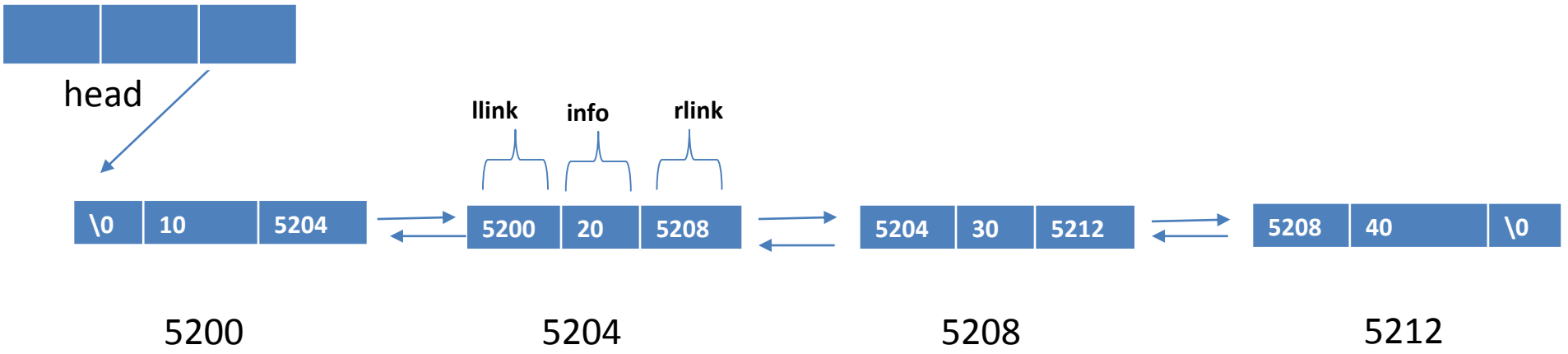




©Ramaiah University of Applied Sciences

# Double linked list

- It is linear collection of nodes where each node consists of
  - info-information is stored
  - llink-pointer field holds the address of left node ( i.e previous node)
  - rlink-pointer field holds the address of right node(i.e next node in list)






©Ramaiah University of Applied Sciences

# Operation on Doubly linked list

- Inserting a node
- Deleting a node
- Display info

```
Struct node
{
    int info;
    struct node *llink;
    struct node *rlink;
};
typedef struct node *NODE;//alias name
```




©Ramaiah University of Applied Sciences

# Inserting Node

```
NODE insert_front(int item,NODE head)
{
    NODE temp,cur;
    temp=getnode();
    temp->info=item;
    cur=head->rlink;
    head->rlink=temp;
    temp->rlink=cur;
    return head;
}
```




©Ramaiah University of Applied Sciences

# Deleting node

```
NODE delete_front(NODE head)
{
    NODE cur,next;
    if(head->rlink==NULL)
{
    printf("list is empty\n");
    return head;
}
    cur=head->rlink;//obtain the first node
    next=cur->rlink;//obtain the second node
    head->rlink=next;
    printf("node deleted is %d",cur->info);
    freenode(cur);//delete the first node
    return head;
}
```


©Ramaiah University of Applied Sciences

# Display

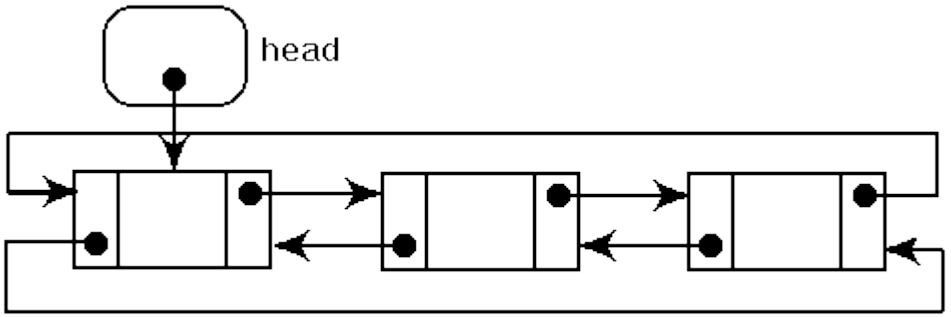
```
Void display(NODE head)
{
    NODE temp;
    if(head->rlink==NULL)
    {
        printf("list is empty");
        return;
    }
    printf("contents of list");
    temp=head->rlink;
    while(temp!=NULL)
    {
        printf("%d",temp->info);
        temp=temp->rlink;
    }
}
```




©Ramaiah University of Applied Sciences

# Circular Doubly Linked List

Circular doubly Linked List is a variation of Linked list in which the first element points to the last element and the last element points to the first element.




Doubly Linked Circular list




©Ramaiah University of Applied Sciences

# Application of linked list

- Arithmetic operations on long positive numbers
- Manipulation of polynomials
- Evaluation of polynomials
- In symbol table construction


©Ramaiah University of Applied Sciences

# Summary

- Linked lists optimise memory usage when compared with arrays for dynamically changing data.



©M. S. Ramaiah University of Applied Sciences