Module 3

CHAPTER 3 Linked List

University Prescribed Syllabus

Introduction, Representation of Linked List, Linked List v/s Array, Types of Linked List - Singly Linked List, Circular Linked List, Doubly Linked List, Operations on Singly Linked List and Doubly Linked List, Stack and Queue using Singly Linked List, Singly Linked List Application-Polynomial Representation and Addition.

Syllabus Topic : Introduction

3.1 INTRODUCTION

- In C programming we have learned the concept of array. Array is a group of elements with same data type.
- Array is considered as an example of static memory allocation. i.e. the size of array is fixed. While declaring an array we have to compulsory mention the size.

Example

```
int arr[5];
```

Element					
Index	0	1	2	3	4

- This is an array *arr* of type *int* and size *5*.
- Size indicates the maximum elements which can be stored in the array.
- In this array we can store maximum of 5 elements.
- But in real life application, it is difficult to predict the maximum elements to be stored as the data is accepted from end user at run time.
- Hence if there is requirement of storing elements more than size of the array, then memory shortage occurs and we cannot store more elements than the size of an array.
- Now in the above array, if only three elements are stored, then the memory will be wasted.

Memory Waste

Element	11	12	13		
Index	0	1	2	3	4

- Here the total memory wastage is 2+2 = 4 bytes.
- Array has both memory shortage as well as memory wastage problems.
- To solve these problems, DS provides the concept of linked list which is an example of **dynamic memory allocation**.

GQ. 3.1.1 *Define dynamic memory allocation. State its importance.* **(2 Marks)**

Definition : Dynamic memory allocation means memory can be allocated or de-allocated as per the requirement at run time.

Importance of dynamic memory allocation

- No need to initially occupy large memory.
- Memory can be allocated as well as de-allocated as per necessity.
- It avoids the memory shortage as well as memory wastage.

3.1.1 Linked List

UQ. 3.1.2 *Explain Linked list as an ADT* **MU - Dec. 15, 2 Marks**

Definition : Linked list is a linear data structure which consists of group of elements called as nodes. Every node contains two parts: data and next. Data contains the actual element while next contains the address of next node.

GQ. 3.1.3 *Draw representation of linear linked list.* **(2 Marks)**

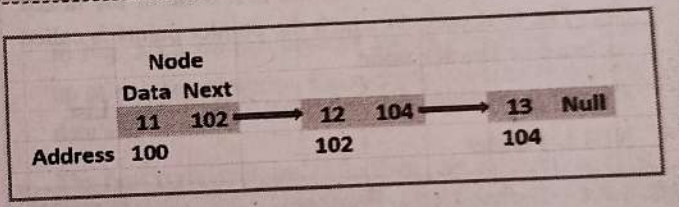


Fig. 3.1.1 : Representation of linear linked list

3.2 TERMINOLOGIES

GQ. 3.2.1 *Define the term node, address, null pointer, next and empty list for linked list.* **(5 Marks)**

GQ. 3.2.2 *Explain the concept of information, next, null pointer and empty list with respect to link list.* **(5 Marks)**

Node

- Every element in a linked list is called as a node. node consists of two parts, Data and Next.
- Data contains the actual element while next contains address of next node.

Information

- Information is nothing but the actual element (record) which is present in the data part of node.
- E.g. roll number or name of student.

Next

GQ. 3.2.3 *Define the term next pointer of linked list.* **(1 Mark)**

- Next is second part of a node. It contains address of next node. It helps to go from one node to other node.

Address

- Next part of every node contains the memory address of next node. This is very useful to go from one node to another i.e. for traversing purpose.
- The last node always contains NULL value in the next part as there is no next node.
- When there is single node, then it contains NULL value in the next part.

Pointer

- A pointer always points to the first node of the linked list. The first node to which the pointer points is called as **head or header node.**
- Pointer is considered as reference to the Linked List.

NULL Pointer

GQ. 3.2.4 *Define NULL pointer.* **(1 Mark)**

- When the linked list is empty, i.e. there is no node in it then NULL value is set to the pointer.

  E.g. P = NULL;

Empty List

- When there is no node in any linked list, then it is called as empty linked list.

3.2.1 Advantages of Linked List

UQ. 3.2.5 *What are the advantages of using linked lists over arrays ?* **MU - May 16, 5 Marks**

UQ. 3.2.6 *State advantages of linked-list over arrays.* **MU - May 18, 5 Marks**

1. Linked lists are dynamic data structures, which can grow or shrink (by allocating or de-allocating memory) while the program is running.
2. Insertion and deletion operations can be easily implemented in a linked list.
3. Dynamic data structures such as stacks and queues can be implemented using a linked list.
4. There is no need to define initial size for a linked list.
5. Items can be added or removed from the middle of the list.
6. Backtracking is possible in two way linked list (Doubly Linked List).

3.2.2 Disadvantages of Linked List

GQ. 3.2.7 *Discuss disadvantages of linked list over array.* **(2 Marks)**

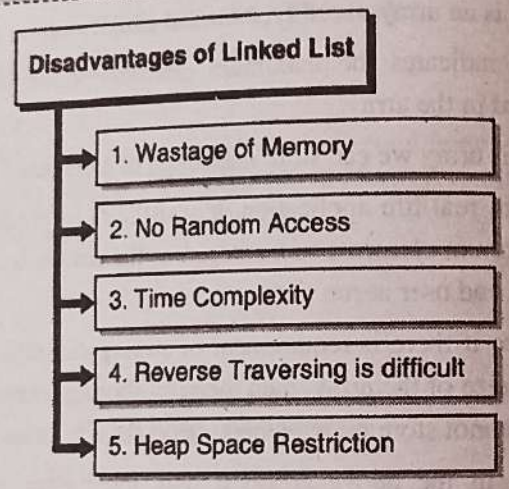


Fig. 3.2.1 : Disadvantages of Linked List

1. Wastage of Memory

- In linked list Pointer needs extra memory for storage.

2. No Random Access

- In array it is possible to access n^{th} element with ease just by using a[n].
- In Linked list there is no random access provided to user, user has to access each node sequentially.
- Just consider that if user wants to access n^{th} node then he has to traverse linked list upto n^{th} node.

3. Time Complexity

- In linked list the nodes are not stored in the **contiguous memory Locations.**
- Hence the access time for Individual node is **O(n)** while in Array it is O(1).

4. Reverse Traversing is difficult

- In case of singly linked list, it is not possible to traverse linked list from end to beginning that means in reverse order.
- With the help of doubly linked list it becomes possible but still it increases the required storage space for back pointer.

5. Heap Space Restriction

- Whenever there is dynamically memory allocation, the memory is utilized from heap.
- Memory is allocated to Linked List at run time if and only if there is space available in heap.
- If there is not sufficient space available in heap then it will not be allocated.

Syllabus Topic : Linked List V/S Array

3.2.3 Differentiate between Array and Linked List

GQ. 3.2.8 *Differentiate between arrays and linked list.* **(4 Marks)**

Parameter	Array	Linked List
Memory allocation	Static memory allocation	Dynamic memory allocation
Memory wastage	Memory is wasted some times	No memory wastage
Allocation	Contiguous	May of may not contiguous
Types	1-D, 2-D, 3-D etc.	Singly, doubly, circular
Stores	Stores elements	Stores nodes as elements

Syllabus Topic : Representation of Linked List

3.3 REPRESENTATION OF LINKED LIST

A linked list can be represented in two ways :

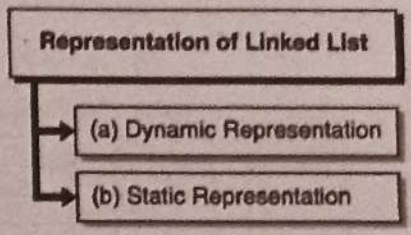


Fig. 3.3.1 : Representation of Linked List

Dynamic Representation of Linked List

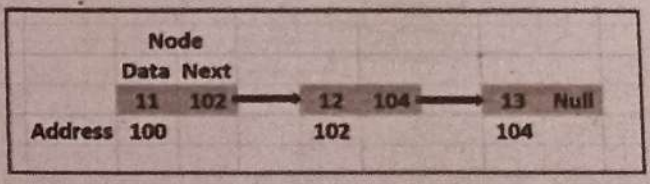


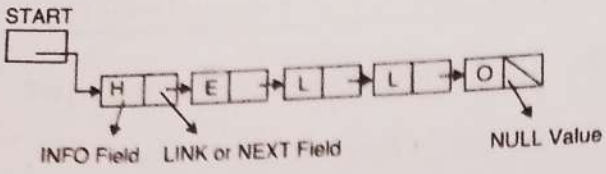
Fig. 3.3.2 : Representation of linear linked list

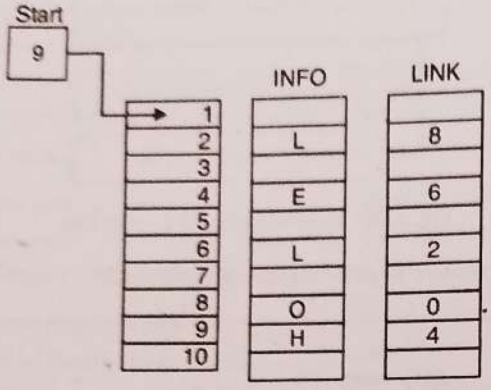
- In Fig. 3.3.2 we can observe that a linked list is made up of number of nodes. Every node contains two parts : data and next.
- The first node contains 11 in data part as value and 102 in next part which is address of next node.
- Every node is connected with next node with the help of address.
- The next part of last node has null value as there is no further node.

2. Static Representation of Linked List

- Let LIST is linear linked list. It needs two linear arrays for memory representation. Let these linear arrays are INFO and LINK.
- INFO[K] contains the information part and LINK[K] contains the next pointer field of node K.
- A variable START is used to store the location of the beginning of the LIST and NULL is used as next pointer sentinel which indicates the end of LIST. It is shown below:

Module 3





Here

START = 9 ⇒ INFO[9] = H is the first character

LINK[9] = 4 ⇒ INFO[4] = E is the second character

LINK[4] = 6 ⇒ INFO[6] = L is the third character

LINK[6] = 2 ⇒ INFO[2] = L is the fourth character

LINK[2] = 8 ⇒ INFO[8] = O is the fifth character

LINK[8] = 0 ⇒ The NULL value, so the LIST ends here.

Syllabus Topic : Types of Linked List

3.4 TYPES OF LINKED LIST

GQ. 3.4.1 *Enlist types of Linked List.* **(3 Marks)**

There are three types of linked lists :

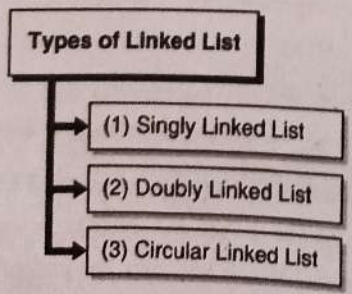


Fig. 3.4.1 : Types of Linked List

Syllabus Topic : Singly Linked List

3.5 SINGLY LINKED LISTS

GQ. 3.5.1 *Explain with suitable diagram : Singly Linked List.* **(2 Marks)**

- **Singly linked list** is the basic form of linked list.
- In this linked list, every node is made up of two parts : data and next. Data part contains the actual element while next part contains address of next node.
- The next part of last node always contains null value.
- It is a one way traversing list, which means we can traverse only in forward direction. We cannot traverse in backward direction.
- The first node is called as **head** node which is considered as reference to the linked list.

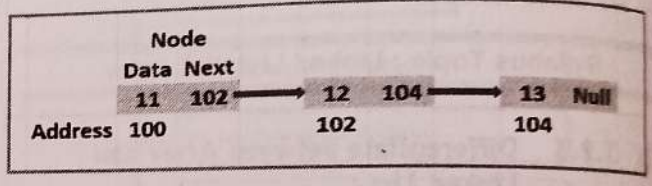


Fig. 3.5.1 : Singly linked list

Syllabus Topic : Operations on Singly Linked List

3.6 OPERATIONS ON SINGLY LINKED LIST

GQ. 3.6.1 *List operations on linked list.* **(2 Marks)**

- There are various types of operations which can be performed on singly linked list :

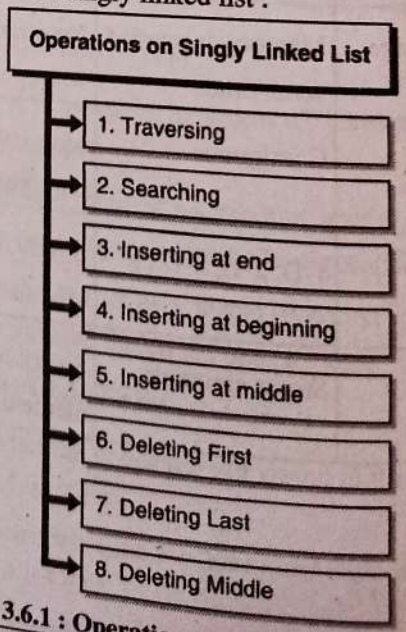


Fig. 3.6.1 : Operations on Singly Linked List

3.6.1 Traversing a Singly Linked List

GQ. 3.6.2 Write an algorithm to traverse a singly linked list. **(1 Mark)**

Traversing means the process of visiting each node at least once.

☞ **Algorithm to traverse singly linked list**

Step 1 : If head = NULL
print ('List is empty')
else
Step 2 : q = head
Step 3 : print (q->data)
Step 4 : go to next node
Step 5 : if q != NULL
Repeat from step 3
Step 6 : End

☞ **C function to traverse and display data**

GQ. 3.6.3 Write C function to traverse and display data. **(2 Marks)**

```
void display ( struct node *q )
{
/* address of head is passed to q */
   while ( q != NULL )
   {
      printf ( "%d ", q -> data ) ;
      q = q -> next ;
   }
}
```

From beginning data part of every node is printed and pointer shifts to next node

3.6.2 Counting Number of Nodes in Singly Linked List

☞ **Algorithm to count number of nodes in singly linked list**

GQ. 3.6.4 Write an algorithm to count number of nodes in singly linked list. **(3 Marks)**

Step 1 : If *head* is NULL then
print ('List is empty')
else
Step 2 : q = head AND counter= 0
Step 3 : counter = counter +1
Step 4 : go to next node
Step 5 : if *q* != NULL
Repeat from step 3
Step 6 : return *counter*

☞ **C function to count number of nodes in Singly Linked List**

GQ. 3.6.5 Write C function to count number of nodes in singly linked list. **(4 Marks)**

```
int count ( struct node *q )
{
int count = 0;
/* address of head is passed to q */
   while ( q != NULL )
   {
      count++;
      q = q -> next ;
   }
   return(count)
}
```

Counter is incremented for every node while traversing

3.6.3 Searching a Linked List

GQ. 3.6.6 Write an algorithm for searching a node in linked list. **(4 Marks)**

– We can **search** for a node in the linked list. The data part can be accepted from user to search. This data part is compared with data parts of all the nodes.

Module 3

☞ **Algorithm to search node in singly linked list**

Step 1 : Read data element to search
Step 2 : If head is NULL then
print ('List is empty')
else
Step 3 : q = head
Step 4 : if data = q->data
print (q->data)
return
Step 5 : go to next node
Step 6 : if q != NULL
Repeat from step 3

☞ **C function to search node in singly linked list**

GQ. 3.6.7 *WAP to search an element in a link list.* **(5 Marks)**

```
void search ( struct node *q, int data )
{
   while ( q != NULL )
   {
    if(data ==q->data)
    {
    printf ( "Element found %d",data) ;
    return;
    }
     q = q -> next ;
  }
  printf("\n Element not found");
}
```

Address of head is passed to q and element to search is assigned to data

3.6.4 Inserting a Node in Singly Linked List

A node can be inserted at any place in the linked list :

(i) At the end,

(ii) At beginning or

(iii) In the middle of linked list

(i) Adding at the end of singly Linked List (append)

☞ **Algorithm to add node at the end of singly linked list**

GQ. 3.6.8 *Write an algorithm to insert new node at the end of a single linked list.* **(7 Marks)**

Step 1 : q = head
Step 2 : Read data
Step 3 : alloc (*newNode*)
newNode→data = data
newNode→next = NULL
Step 4 : If *q* is NULL then
q = newNode and return
Step 5 : *temp* = *q*
Step 6 : go to next node
Step 7 : if temp->next is not NULL then
Repeat Step 6
Step 8 : Set address of newnode in next of temp

☞ **Representation**

– When first node is added (i.e. when linked list is empty)

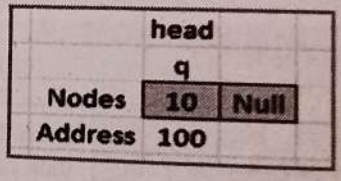


Fig. 3.6.2 : Adding first node

– When already nodes are present :

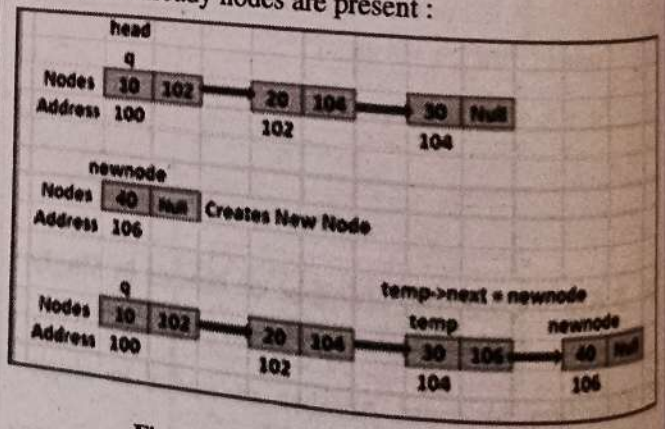


Fig. 3.6.3 : Adding node in existing list

☞ **C function to add node at the end of singly linked list**

GQ. 3.6.9 *Write a 'C' function to insert a new node at the end into singly linked list.* **(3 Marks)**

```
void append ( struct node **q, int num )
{
    struct node *temp, *newnode ;

    newnode = malloc ( sizeof ( struct node ) ) ;
    newnode -> data = num ;
    newnode -> next = NULL ;

if ( *q == NULL )
        {
        *q = newnode ;
        }
 else
    {
        temp = *q ;
        while ( temp -> next != NULL )
            temp = temp -> next;

        temp -> next = newnode ;
    }
}
```

if the list is empty, create first node

Go to last node

(ii) Adding node at the beginning of Singly Linked List

☞ **Algorithm to add node at the beginning of singly linked list**

GQ. 3.6.10 *Write an algorithm to insert new node at the beginning of a single linked list.* **(2 Marks)**

Step 1 : set q at head
Step 2 : Read data
Step 3 : alloc (newNode)
newNode→data = data
Step 4 : newNode→next = q
Step 5 : Set q at newnode

☞ **Representation**

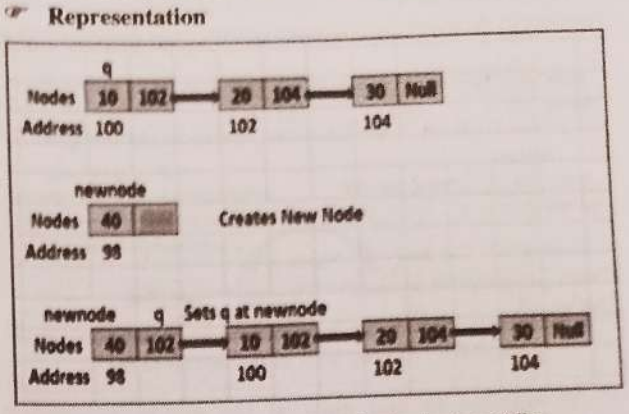


Fig. 3.6.4 : Adding node at the beginning of list

☞ **C function to add node at the beginning of linked list**

GQ. 3.6.11 *Write C function to add node at the beginning of linked list.* **(4 Marks)**

```
void addatbeg ( struct node **q, int num )
{
    struct node *temp, *newnode ;
    newnode = malloc ( sizeof ( struct node ) ) ;
    newnode -> data = num ;
    newnode -> next = *q;
    *q = newnode;
}
```

The address of previous first node is assigned to next part of new node which makes new node as first node

(iii) Adding at the middle

☞ **Algorithm to add node at the middle of singly linked list**

Step 1 : temp = head;
Step 2 : read num
Step 3 : traverse temp to location
Step 4 : alloc (newNode)
Step 5 : newNode → data = num
Link newnode with next node
Link newnode with previous node

☞ Representation

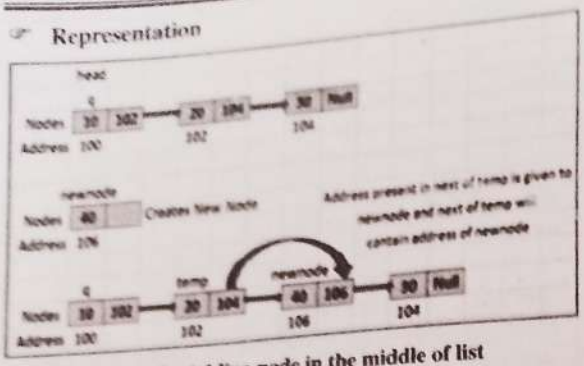


Fig. 3.6.5 : Adding node in the middle of list

☞ C function to add node at the middle of singly linked list

GQ. 3.6.12 *Write C function to add node at the middle of singly linked list.* **(4 Marks)**

```
void addafter ( struct node *q, int num, int loc)
{
    struct node *temp, *r, *newnode ;
    int i ;

    temp = q ;

    for ( i = 0 ; i < loc-1 ; i++ )
    {
        temp = temp -> next ;
    }
    /* if end of linked list is encountered */
    if ( temp == NULL )
    {
        printf ( "\n Location is wrong", loc ) ;
        return ;
    }
    /* insert new node */
    newnode = malloc ( sizeof ( struct node ) ) ;
    newnode -> data = num ;
    newnode -> next = temp -> next ;
    temp -> next = newnode ;
}
```

Temp traversed up to location

Setting node at the middle by exchanging addresses

3.6.5 Deleting a Node from Singly Linked List

GQ. 3.6.13 *Write an algorithm to delete an element from a singly linked list.* **(7 Marks)**

Step 1 : q = head;

Step 2 : temp = head;

Step 3 : traverse q to next node

Step 4 : delete temp

☞ Representation

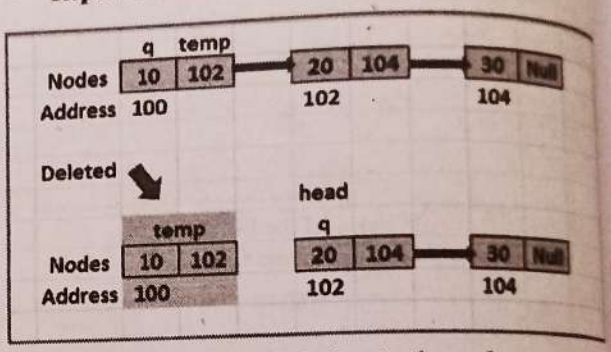


Fig. 3.6.6 : Deleting starting node

☞ C function to delete node which is at the beginning of linked list

UQ. 3.6.14 *Write a function for deletion of a node from linked list.* MU - Dec. 15, 8 Marks

GQ. 3.6.15 *Write C function to delete node which is at the beginning of linked list.* **(4 Marks)**

```
void deletefirst (struct node **q)
{
    struct node *temp, *r ;

    temp = *q ;
    *q = temp->next;
    free(temp);
}
```

Sets *q* pointer to second node

(ii) Deleting last node from singly linked list

☞ **Algorithm to delete node which is at the last in the singly linked list**

GQ. 3.6.16 *Explain how to delete last node in linked list.* **(4 Marks)**

Step 1 : q = head;
Step 2 : temp = head;
Step 3 : alloc(old)
Step 4 : old = temp;
temp = temp→next
Step 5 : if temp→next != NULL repeat step 4
Step 6 : Set NULL value in old->next
Step 7 : Delete temp

☞ **Representation**

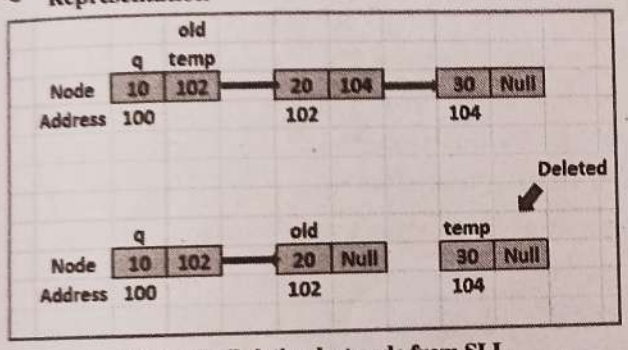


Fig. 3.6.7 : Deleting last node from SLL

☞ **C function to delete node which is at the last in linked list**

GQ. 3.6.17 *Write C function to delete node which is at the last in the singly linked list.* **(4 Marks)**

```
void deletelast (struct node **q)
{
    struct node *temp,*old;
    temp = *q;
while (temp->next != NULL)
{
      old = temp;
      temp = temp->next;
}
old -> next = temp -> next ;
free(temp);
}
```

temp is set to last node and deleted

(iii) Deleting node from middle of singly linked list.

Here we accept data of node to be deleted from user.

☞ **Algorithm to delete node which is at the middle in the linked list**

GQ. 3.6.18 *Write Algorithm to delete node which is at the middle in the linked list.* **(4 Marks)**

Step 1 : q = head;
Step 2 : temp = head;
Step 3 : Read num : data of node to be deleted
Step 4 : alloc(old)
Step 5 : If temp->data = num
Link previous and next node of temp
Else
Go to next node and set old = temp
Step 6 : if temp is not NULL
Repeat from step 5
Step 7 : Delete temp

☞ **Representation**

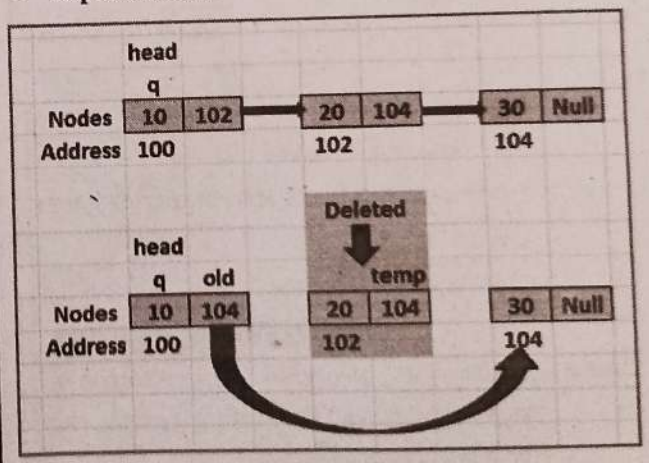


Fig. 3.6.8 : Deleting middle node

☞ **C function to delete node which is at the middle in singly linked list**

GQ. 3.6.19 *Write C function to delete node which is at the middle in singly linked list.* **(4 Marks)**

Module 3

```
void deletemiddle (struct node **q, int num)
{
struct node *temp,*old;
temp = *q;

while (temp->next != NULL)
{
    if(temp->data == num)
    {
        old -> next = temp -> next;
        free(temp);
        return;
    }
    else
    old = temp;

    temp = temp->next;
    }
printf("\n Element not found");
}
```

When node found, its previous and next nodes are linked. And then the node is deleted

3.6.6 Menu Driven Program on All Operations of Singly Linked List

UQ. 3.6.20 Write a 'C' program to create a "Single Linked List" ADT. The ADT should support the following functions :

(i) Creating a Linked List

(ii) Inserting a node after a specific node

(iii) Deleting a node

(iv) Displaying the list

MU - Dec. 13, Dec. 16, 12 Marks

UQ. 3.6.21 Write a 'C' program to implement a singly Linked List which supports the following operations :

(i) Insert a node in the beginning

(ii) Insert a node in the end

(iii) Insert a node after a specific node

(iv) Deleting a specific node

(v) Displaying the list.

MU - Dec. 14, May 17, 10 Marks

UQ. 3.6.22 Write a program to implement singly linked list. Provide the following operations :

(i) Insert a node at the specified location.

(ii) Delete a node from end

(iii) Display the list

MU - May 18, Dec.18, 10 Marks

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
void display ( struct node *q );
void search ( struct node *q, int data );
void append ( struct node **q, int num );
void addatbeg ( struct node **q, int num );
void addafter ( struct node *q, int num, int loc );
void deletefirst (struct node **q);
void deletelast (struct node **q);
void deletemiddle (struct node **q, int num);

struct node
{
    int data ;
    struct node *next ;
} ;

main()
{
struct node *head ;
int ch=0,ele,loc;
    head = NULL ;
    clrscr();
    while(ch!=9)
    {
    printf("\n 1 : Append");
    printf("\n 2 : Display");
    printf("\n 3 : Search");
    printf("\n 4 : Addafter");
```

Structure containing data part and link part

Empty linked list

```
printf("\n 5 : Addatbeg");
printf("\n 6 : DeleteFirst");
printf("\n 7 : DeleteLast");
printf("\n 8 : DeleteMiddle");
printf("\n 9 : Exit");

printf("\n Select your choice : ");
scanf("%d",&ch);

switch(ch)
{
case 1:
printf("\n Enter element to append : ");
scanf("%d",&ele);
append(&head,ele);
break;
case 2:
display(head);
break;
case 3:
printf("\n Enter element to search : ");
scanf("%d",&ele);
search(head,ele);
break;

case 4:
printf("\n Enter element and location : ");
scanf("%d %d",&ele,&loc);
addafter(head,ele,loc);
break;

case 5:
printf("\n Enter element to add at beginning : ");
scanf("%d",&ele);
addatbeg(&head,ele);
break;

case 6:
deletefirst(&head);
break;

case 7:
deletelast(&head);
break;

case 8:
printf("\n Enter element to delete : ");
scanf("%d",&ele);
deletemiddle(&head,ele);
break;

default:
printf("\n Invalid choice");
}
}
}

void append ( struct node **q, int num )
{
    struct node *temp, *newnode ;

newnode = malloc ( sizeof ( struct node ) ) ;
    newnode -> data = num ;
        newnode -> next = NULL ;

if ( *q == NULL )
    {
        *q = newnode ;
        }
else
    {
        temp = *q ;

        /* go to last node */
        while ( temp -> next != NULL )
            temp = temp -> next;
```

If the list is empty, create first node

Go to last

```
        temp -> next = newnode ;
    }
}

void display ( struct node *q )
{
    while ( q != NULL )
    {
        printf ( "%d ", q -> data ) ;
        q = q -> next ;
    }
}
```

From beginning data part of every node is printed and pointer shifts to next node

```
void search ( struct node *q, int data )
{
    while ( q != NULL )
    {
     if(data ==q->data)
      {
      printf ( "Element found %d",data) ;
      return;
      }
       q = q -> next ;
    }
    printf("\n Element not found");
}
```

Address of head is passed to q and element to search is assigned to data

```
void addatbeg ( struct node **q, int num )
{
    struct node *temp, *newnode ;

newnode = malloc ( sizeof ( struct node ) ) ;
    newnode -> data = num ;
    newnode -> next = *q;
*q = newnode;
}
```

The address of previous first node is assigned to next part of new node which makes new node as first node

```
void addafter ( struct node *q, int num, int loc )
{
    struct node *temp, *r, *newnode ;
    int i ;

    temp = q ;

    for ( i = 0 ; i < loc-1 ; i++ )
    {
        temp = temp -> next ;

        if ( temp == NULL )
        {
            printf ( "\n Location is wrong", loc ) ;
            return ;
        }
    }

    /* insert new node */
    newnode = malloc ( sizeof ( struct node ) ) ;
    newnode -> data = num ;
    newnode -> next = temp -> next ;
    temp -> next = newnode ;
}
void deletefirst (struct node **q)
{
    struct node *temp, *r ;

    temp = *q ;
    *q = temp->next;
    free(temp);
}
void deletelast (struct node **q)
{
    struct node *temp,*old;
    temp = *q;
```

Temp traversed up to location

Sets q pointer to second node

```
while (temp->next != NULL)


{
      old = temp;
      temp = temp->next;
}
old -> next = temp -> next ;
free(temp);
}

void deletemiddle (struct node **q, int num)
{
struct node *temp,*old;
temp = *q;

while (temp->next != NULL)
{
if(temp->data == num)
{
old -> next = temp -> next;
free(temp);
return;
}
else
old = temp;

temp = temp->next;
}
printf("\n Element not found");

}
```

Temp is set to last node and deleted

When node found, its previous and next nodes are linked. And then the node is deleted

Output

```
1 : Append
2 : Display
3 : Search
4 : Addafter
5 : Addatbeg
6 : DeleteFirst
7 : DeleteLast
8 : DeleteMiddle
9 : Exit
Select your choice : 1

Enter element to append : 11

1 : Append
2 : Display
3 : Search
4 : Addafter
5 : Addatbeg
6 : DeleteFirst
7 : DeleteLast
8 : DeleteMiddle
9 : Exit
Select your choice :
```

Module 3

Syllabus Topic : Stack using Singly Linked List / Linked Implementation of Stack

3.7 STACK USING SINGLY LINKED LIST / LINKED IMPLEMENTATION OF STACK

UQ. 3.7.1 Write the implementation procedure of basic primitive operations of the stack using Linked list. MU - Dec. 09, 5 Marks

UQ. 3.7.2 Write a program to implement stack using linked list. MU - May. 16, Dec. 16, 7 Marks

UQ. 3.7.3 Write a program in 'C' to implement Stack using Linked-List. Perform the following operations :

(i) Push (ii) Pop (iii) Peek

(iv) Display the stack contents. MU - May 19, 10 Marks

– As we have seen, in stack elements are placed one above other. In the same manner in *stack as linked list*, we will place nodes one above other.

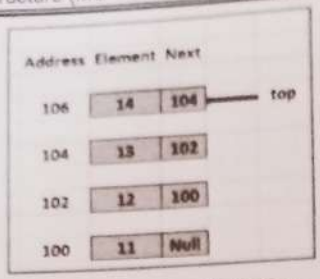


Fig. 3.7.1 : Stack as Linked List

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>

struct node
{
    int data ;
    struct node *next;
} ;

void push ( struct node **, int ) ;
void  pop ( struct node ** ) ;
void display( struct node * ) ;

void main( )
{
    struct node *head = NULL ;
    int i,ch,ele ;

    clrscr( ) ;

while(ch!=4)
{
printf("\n\t 1: Push");
printf("\n\t 2: Pop");
printf("\n\t 3: Display");
printf("\n\t 4: Exit");
printf("\n Select your choice : ");
scanf("%d",&ch);

switch(ch)
{
case 1:
printf("\n Enter element to insert : ");
scanf("%d",&ele);
push(&head,ele);
break;

case 2:
pop(&head);
break;
case 3:
display(head);
break;

case 4:
exit(0);
break;

deault:
printf("\n Invalid choice");
}
}
    getch( ) ;
}

void push ( struct node **top, int ele )
{
    struct node *temp ;
    temp = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
    if ( temp == NULL )
        printf ( "\nStack is Overflow" ) ;
    temp -> data = ele ;
    temp -> next = *top ;
    *top = temp ;
}
void  pop ( struct node **top )
{
    struct node *temp ;
    int item ;
```

Structure containing a data part and link part

Empty linked list

As per user's choice operation will be performed

Store address of top in temp and set top to temp

```
    if ( *top == NULL )
    {
        printf ( "\nStack is Underflow" ) ;
        return NULL ;
    }
    temp = *top ;
    item = temp -> data ;
    *top = ( *top ) -> next ;
    free ( temp ) ;
printf("\n Element popped : %d",item);
}

void display ( struct node *top )
{
    struct node *temp ;

    if ( top == NULL )
        return ;

    temp = top;

    while ( temp != NULL )
    {
        printf(" %d ",temp->data);
        temp = temp->next;
    }
}
```

Set temp at top. Traverse top to next node and delete temp

Display data part starting from top

Output

```
C:\LINKS.exe

        1: Push
        2: Pop
        3: Display
        4: Exit
Select your choice : 1
```

Syllabus Topic : Queue using Singly Linked List / Circular Linked List

3.8 QUEUE USING SINGLY LINKED LIST / CIRCULAR LINKED LIST

GQ. 3.8.1 *Explain the term : Circular linked list.* **(5 Marks)**

- In a singly linked list, the traversing is allowed in forward direction only. In this list, the next part of last node contains NULL value.
- Just consider there are 100 nodes. Currently we are at node number 98 and we want to go to node number 3, then it is not possible.
- DS solves this issue by providing concept of **Circular Linked List** in which the next part of last node contains address of first node.
- Hence we can directly jump from last node to first node.

Module 3

❑ *Definition : Circular linked list is the linked list in which next part of last node contains address of first node to form a circle.*

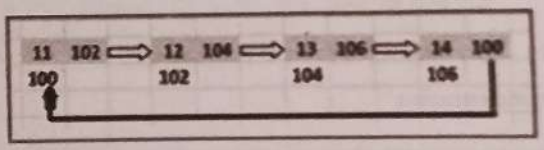


Fig. 3.8.1 : Circular linked list

- In circular linked list, we can use the concept of Queue. Hence it can also be considered as **Queue as Linked List.**
- You may remember that Queue is a data structure in which addition of node is allowed at one end called as *rear* while removal of node is allowed at another end called as *front*.

3.8.1 Operations on Circular Linked List

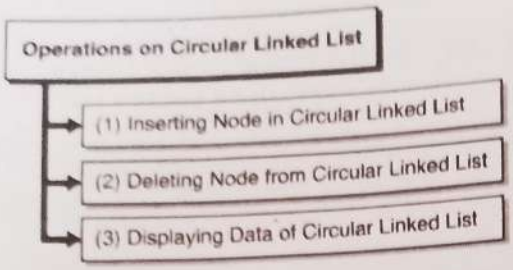


Fig. 3.8.2 : Operations on Circular Linked List

(1) Inserting Node in Circular Linked List

Algorithm to insert node in circular linked list

GQ. 3.8.2 Write algorithm to insert node in circular linked list. **(4 Marks)**

Step 1 : alloc (*newNode*)
Step 2 : read data for node
Step 3 : If rear is NULL
Set both front and rear to newnode
Else
Assign address of newnode to rear->next
Set rear to newnode
Step 4 : Set address of front in rear->next

Representation

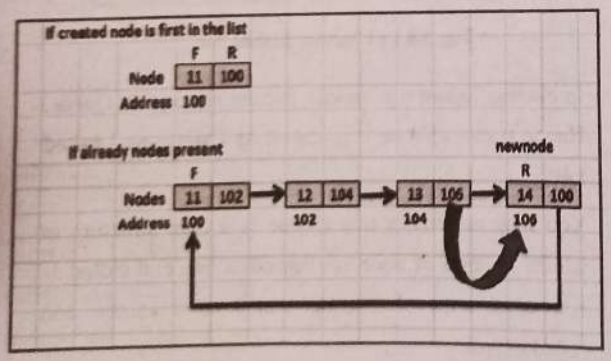


Fig. 3.8.3 : Inserting node in circular linked list

Function to insert node in circular linked list

GQ. 3.8.3 Write C function to insert node in circular linked list. **(5 Marks)**

```
void insert()
{
  node *newnode;
  newnode=(node*)malloc(sizeof(node));
  printf("\nEnter the node value : ");
  scanf("%d",&newnode->data);
  newnode->next=NULL;
  if(rear==NULL)
  front=rear=newnode;
  else
  {
    rear->next=newnode;
    rear=newnode;
  }
  rear->next=front;
}
```

If new node is first node

Address of newnode is assigned to rear.next and newnode becomes new rear

Address of front is assigned to new rear

(2) Deleting Node from Circular Linked List

Algorithm to delete node from circular linked list

GQ. 3.8.4 Write algorithm to delete node from circular linked list. **(4 Marks)**

Step 1 : temp=front;
Step 2 : if front is NULL
print "Underflow"
else
if(front==rear) i.e. last one is deleted
set front and rear to NULL;
else
set front to next node
Step 3 : assign address of new front to rear
Step 4 : Delete (temp)

☞ **Representation**

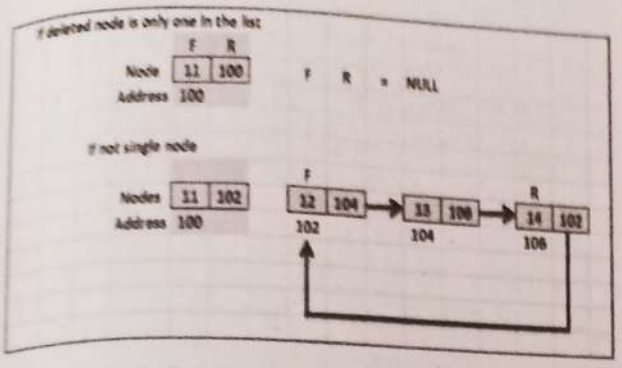


Fig. 3.8.4 : Deleting node from circular linked list

☞ **Function to delete node from circular linked list**

GQ. 3.8.5 *Write C function to delete node from circular linked list :* **(4 Marks)**

```
void del()
{
  temp=front;                          <- If list is empty
  if(front==NULL)
    printf("\nUnderflow...");
  else
  {
    if(front==rear)                    <- If last node is deleted
    {
      front=rear=NULL;
    }
    else
    {
      front=front->next;               <- Traverse front to next node and assign address of new front to rear
      rear->next=front;
    }
  printf("\n Node deleted : %d",front->data);
  temp->next=NULL;
  free(temp);                          <- Delete temp
  }
}
```

▶ **(3) Displaying Data of Circular Linked List**

☞ **Algorithm to display data of nodes in circular linked list**

GQ. 3.8.6 *Write algorithm to display data from circular linked list.* **(4 Marks)**

Step 1 : Set temp to front;

Step 2 : if front is NULL
print "Linked list is empty"

Step 3 : print temp-data
Go to next node

Step 4 : if temp!=rear
Repeat step 3

☞ **Function to display data of nodes in circular linked list**

GQ. 3.8.7 *Write C function to display data from circular linked list.* **(4 Marks)**

```
void display()
{
  temp=front;
  if(front==NULL)
    printf("\nLinked list is empty");
  else
  {
    printf("\n");
    for(;temp!=rear;temp=temp->next)
    {
      printf(" %d -> ",temp->data);
    }
    printf(" %d ",temp->data);
  }
}
```

Module 3

3.8.2 Program to Implement Queue using Singly Linked List

UQ. 3.8.8 Write a program in 'C' to implement circular queue using Link-list.

MU - May 14, 10 Marks

UQ. 3.8.9 Write a program in 'C' to implement QUEUE ADT using linked-list. Perform the following operations :

(i) Insert a node in the queue.

(ii) Delete a node from the queue.

(iii) Display queue elements.

MU - May 18, 10 Marks

UQ. 3.8.10 Write a program to implement Circular Linked List. Provide the following operations :

(i) Insert a node.

(ii) Delete a node

(iii) Display the list

MU - May 19, Dec.19, 10 Marks

```
#include<stdio.h>
#include<stdlib.h>

typedef struct Node

{
   int data;
   struct Node *next;
}node;

node *front=NULL,*rear=NULL,*temp;

void insert();
void del();
void display();

int main()
{
   int ch;
   clrscr();
   do
   {
   printf("\n\t....Menu.....");
   printf("\n\t 1 : Insert node");
   printf("\n\t 2 : Delete node");
   printf("\n\t 3 : Display");
   printf("\n\t 4 : Exit");
   printf("\nSelect your choice : ");

   scanf("%d",&ch);

    switch(ch)
    {
       case 1:
       insert();
       break;

       case 2:
       del();
       break;

       case 3:
       display();
       break;

       case 4:
       return 1;

       default:
          printf("\nInvalid choice :");
    }
   }while(1);
   return 0;
}
void insert()
{
   node *newnode;
   newnode=(node*)malloc(sizeof(node));
```

Node Structure

Function prototypes

As per user's choice operation will be performed

Creating new node

```
printf("\nEnter the node value : ");
scanf("%d",&newnode->data);
newnode->next=NULL;

if(rear==NULL)
front=rear=newnode;
else
{
rear->next=newnode;
rear=newnode;
}
rear->next=front;
}
```

If new node is first node

Address of newnode is assigned to rear.next and newnode becomes new rear

Address of front is assigned to new rear

```
void del()
{
temp=front;
if(front==NULL)
  printf("\nUnderflow...");
else
{
  if(front==rear)
  {
    front=rear=NULL;
  }
  else
  {
    front=front->next;
    rear->next=front;
  }
printf("\n Node deleted : %d",front->data);
temp->next=NULL;
free(temp);
}
}
```

If list is empty

If last node is deleted

Traverse front to next node and assign address of new front to rear

Delete temp

```
void display()
{
temp=front;
if(front==NULL)
  printf("\nLinked list is empty");
else
{
  printf("\n");
  for(;temp!=rear;temp=temp->next)
  {
    printf(" %d -> ",temp->data);
  }
  printf(" %d ",temp->data);
}
}
```

Display elements from front to rear

Output

```
C:\LINK3.exe

        ....Menu.....
         1 : Insert node
         2 : Delete node
         3 : Display
         4 : Exit
Select your choice : 1

Enter the node value : 11

        ....Menu.....
         1 : Insert node
         2 : Delete node
         3 : Display
         4 : Exit
Select your choice : 1

Enter the node value : 12

        ....Menu.....
         1 : Insert node
         2 : Delete node
         3 : Display
         4 : Exit
Select your choice : 3

 11 ->  12
```

Syllabus Topic : Doubly Linked List

3.9 DOUBLY LINKED LIST

UQ. 3.9.1 What is a doubly linked list ? Give C representation for the same.
MU - Dec. 18, 5 Marks

- In singular linked list, we have seen that traversing is allowed only in one direction i.e. forward direction. We cannot traverse in backward direction.
- Hence from any node if there is need to go to previous node, then it is not possible.
- DS solve this problem by providing the concept of doubly linked list.

Definition : Doubly linked list is the list in which every node contains three parts : data, previous and next. Data contains the actual element while the previous and next parts contain the addresses of previous and next nodes.

Representation of doubly linked list

GQ. 3.9.2 Draw representation of doubly linked list. (2 Marks)

- As every node contains address of previous node along with next node, it is possible to traverse in both the directions. Forward as well as backward.

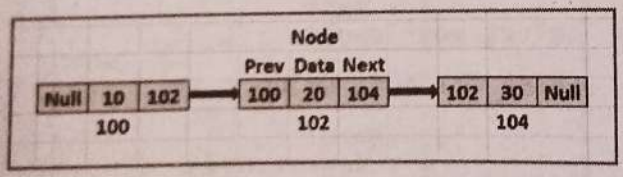


Fig. 3.9.1 : Representation of doubly linked list

- The previous part of first node and next part of last node always contains NULL value.

3.9.1 Advantages of Doubly Linked List over Singly List

GQ. 3.9.3 What are the advantages of doubly linked list? (2 Marks)

GQ. 3.9.4 Briefly explain advantages of doubly link list over single link list. (3 Marks)

1. Doubly Linked List provides two pointers : next and previous which contain addresses of next and previous nodes.
2. A Doubly Linked List can be traversed in both forward as well as backward directions.
3. The delete operation in Doubly Linked List is considered as more efficient if pointer to the node to be deleted is given.

Syllabus Topic : Operations on Doubly Linked List

3.9.2 Operations on Doubly Linked List

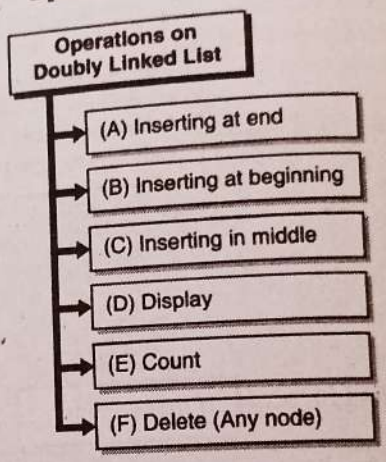


Fig. 3.9.2 : Operations on Doubly Linked List

(A) Inserting node at the end of DLL

Algorithm to add node at the end of DLL

GQ. 3.9.5 Write algorithm to add node at the end of DLL. (4 Marks)

Step 1 : q = head
Step 2 : If q is NULL then
q = newNode and return
Step 3 : temp = q
Step 4 : go to next node
Step 5 : if temp->next is not NULL then
Repeat Step 4
Step 6 : alloc(r)
r->data = data
r->next - NULL
Step 7 : Set address of r in next of temp
Step 8 : Set address of temp in prev of r

Representation

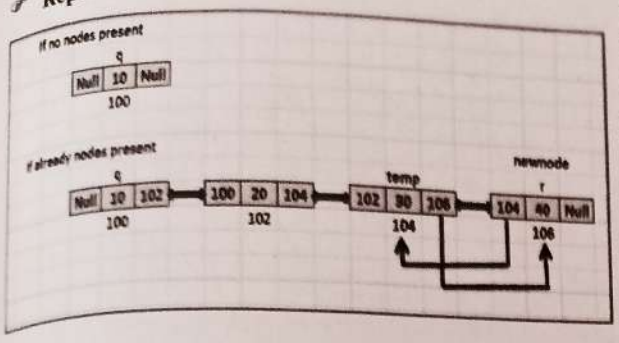


Fig. 3.9.3 : Adding node at the end of DLL

Function to add node at the end of DLL

GQ. 3.9.6 *Write C function to add node at the end of DLL.* **(4 Marks)**

```
void append ( struct node **q, int num )
{
   struct node *r, *temp = *q ;

   if ( *q == NULL )                    <- Creating first node
   {
      *q = malloc ( sizeof ( struct node ) ) ;
      ( *q ) -> prev = NULL ;
      ( *q ) -> data = num ;
      ( *q ) -> next = NULL ;
   }
   else
   {
      while ( temp -> next != NULL )
         temp = temp -> next ;

      r = malloc ( sizeof ( struct node ) ) ;
      r -> data = num ;
      r -> next = NULL ;                <- temp is set to last node. New node r is created and its address is stored in temp->next
      r -> prev = temp ;
      temp -> next = r ;
   }
}
```

(B) Inserting node at the beginning of DLL

Algorithm to add node at the beginning of DLL

GQ. 3.9.7 *Write algorithm to add node at the beginning of DLL.* **(4 Marks)**

Step 1 : q = head

Step 2 : alloc (*temp*)

Step 2 : temp->next = q

Step 2 : temp->prev= NULL

Step 4 : set q to temp

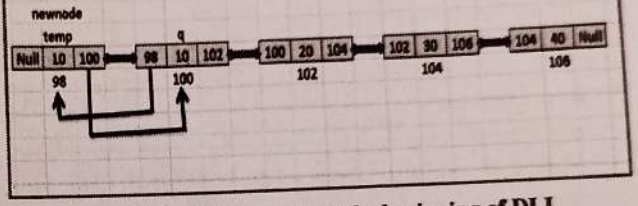


Fig. 3.9.4 : Adding node at the beginning of DLL

Function to add node at the beginning of DLL

GQ. 3.9.8 *Write C function to add node at the beginning of DLL.* **(4 Marks)**

```
void addatbeg ( struct node **q, int num )
{
   struct node *temp ;
   temp = malloc ( sizeof ( struct node ) ) ;
   temp -> prev = NULL ;
   temp -> data = num ;
   temp -> next = *q ;
   ( *q ) -> prev = temp ;
   *q = temp ;
}
```

Module 3

(C) Inserting node at the middle of DLL

Algorithm to add node in the middle of DLL

GQ. 3.9.9 *Write Algorithm to add node in the middle of DLL.* **(3 Marks)**

Step 1 : q = head

Step 2 : Read element and location

Step 3 : Set q to node after which new node has to be added

Step 4 : alloc (temp)
temp →data = data
set address of q in temp->prev
temp->next = q->next
set address of temp in q->next

Representation

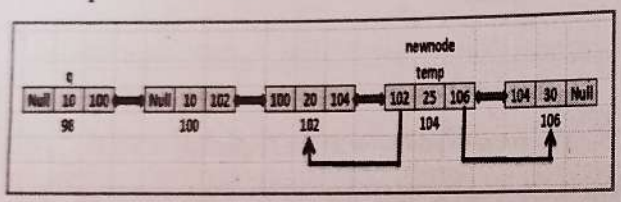


Fig. 3.9.5 : Adding node in the middle of DLL

Function to add node in the middle of DLL

GQ. 3.9.10 *Write C function to add node in the middle of DLL.* **(3 Marks)**

```
void addafter ( struct node *q, int ele, int loc )
{
    struct node *temp ;
    int i ;

    for ( i = 0 ; i < loc ; i++ )
    {
        q = q -> next ;
        if ( q == NULL )
        {
            printf ( "\nInvalid location %d", loc );
            return ;
        }
    }
    q = q -> prev ;
    temp = malloc ( sizeof ( struct node ) ) ;
    temp -> data = ele ;
    temp -> prev = q ;
    temp -> next = q -> next ;
    temp -> next -> prev = temp ;
    q -> next = temp ;
}
void display ( struct node *q )
{
    while ( q != NULL )
    {
        printf ( " <-%d -> ", q -> data ) ;
        q = q -> next ;
    }
}
```

Set q to node after which new node has to be added

Sets new node *temp*. *Temp*'s previous and next nodes are linked to *temp*.

(D) Displaying doubly linked list

Algorithm to display doubly linked list

GQ. 3.9.11 *Write algorithm to display data from DLL.* **(3 Marks)**

Step 1 : If *head* is NULL then
print ('List is empty')
else

Step 2 : *q = head*

Step 3 : print (*q->data*)

Step 4 : go to next node

Step 5 : if *q* != NULL
Repeat from step 3

Function to display doubly linked list

GQ. 3.9.12 *Write C function to display data from DLL.* **(4 Marks)**

GQ. 3.9.13 *Write 'C' functions to implement TRAVERSE operation in doubly linked list.* **(2 Marks)**

```
void display ( struct node *q )
{
   while ( q != NULL )
   {
      printf ( " <-%d -> ", q -> data ) ;
      q = q -> next ;
   }
}
```

(E) Counting nodes of Doubly Linked List

Algorithm to count nodes of Doubly Linked List

GQ. 3.9.14 Write Algorithm to count nodes of Doubly Linked List **(4 Marks)**

Step 1 : If *head* is NULL then
print ('List is empty')
else

Step 2 : q = head AND counter= 0

Step 3 : counter = counter +1

Step 4 : go to next node

Step 5: if *q* != NULL
Repeat from step 3

Step 6 : return *counter*

Function to count nodes of Doubly Linked List

GQ. 3.9.15 Write a C function to find maximum element from doubly linked list. **(7 Marks)**

```
int count ( struct node * q )
{
   int cnt = 0 ;
   while ( q != NULL )
   {
      q = q -> next ;
      cnt++ ;
   }
   return cnt ;
}
```

(F) Deleting node from Doubly Linked List

GQ. 3.9.16 Explain delete operation of doubly linked list. **(7 Marks)**

Algorithm to delete node from Doubly Linked List

GQ. 3.9.17 Write algorithm to delete node from Doubly Linked List. **(4 Marks)**

q = temp = head

Step 2 : Read num

Step 3 : Traverse temp

Step 4 : Compare temp->data with num

Step 5 : If data found :

If it is first node, set q to next node

Set NULL value in q->prev

If it is last node, Set NULL value in next of temp->prev node

If it is intermediated node

Link the previous and next node of temp

Step 6 : Delete temp

Step 7 : If data not found and temp!= NULL

Repeat from Step 3

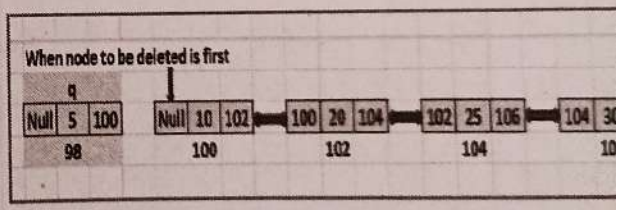


Fig. 3.9.6 : Deleting first node from doubly linked lis

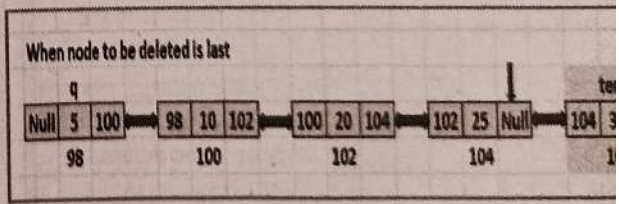


Fig. 3.9.7 : Deleting last node from doubly linked lis

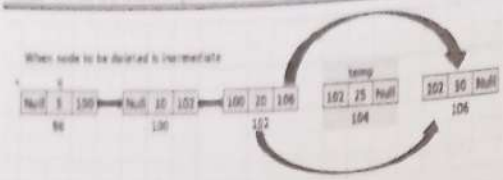


Fig. 3.9.8 : Deleting intermediate node from doubly linked list

☞ **Function to delete node from Doubly Linked List**

GQ. 3.9.18 *Write C function to delete node from Doubly Linked List.* **(4 Marks)**

GQ. 3.9.19 *Write 'C' functions to implement DELETE_FIRST_NODE operation in doubly linked list.* **(4 Marks)**

```
void delete ( struct node **q, int num )
{
    struct node *temp = *q ;
    while ( temp != NULL )
    {
        if ( temp -> data == num )
        {
            if ( temp == *q )          <- if node to be deleted is the first node
            {
                *q = ( *q ) -> next ;
                ( *q ) -> prev = NULL ;
            }
            else
            {
if ( temp -> next == NULL )            <- if node to be deleted is the last node
temp -> prev -> next = NULL ;
            else                       <- if node to be deleted is intermediate node
            {
                temp -> prev -> next = temp -> next ;
                temp -> next -> prev = temp -> prev ;
                }
                free ( temp ) ;
                printf("\n Node deleted %d",num);
            }
            return ;
        }
        temp = temp -> next ;
    }
    printf ( "\n%d not found.", num ) ;
}
```

☞ **Program to perform various operations on doubly linked list**

UQ. 3.9.20 *Write a program in 'C' to implement Doubly Link-list with methods insert, delete and display.*

MU - May 14, May 17, 10 Marks

GQ. 3.9.21 *Write a C program to implement a Doubly Linked List which performs the following operations :*

(i) Inserting element in the beginning
(ii) Inserting element in the end
(iii) Inserting element after an element
(iv) Deleting a particular element
(v) Displaying the list **(12 Marks)**

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
struct node
{
    struct node *prev ;
    int data ;
    struct node * next ;
} ;
void append ( struct node **, int ) ;
void addatbeg ( struct node **, int ) ;
void addafter ( struct node *, int , int ) ;
void display ( struct node * ) ;
int count ( struct node * ) ;
void delete ( struct node **, int ) ;
```

```
void main( )
{
  struct node *head ;
  char ch;
  int ele,loc;
  head = NULL ;
while(ch'= 7)
  {
  printf("\n\n 1 : Append");
  printf("\n 2 : Addatbeg");
  printf("\n 3 : Addafter");
  printf("\n 4 : Display");
  printf("\n 5 : Count");
  printf("\n 6 : Delete");
  printf("\n 7 : Exit");
  printf("\n Select your choice : ");
  scanf("%d",&ch);
  switch(ch)
  {
  case 1:
  printf("\n Enter element to append : ");
  scanf("%d",&ele);
  append(&head,ele);
  break;
  case 2:
  printf("\n Enter element to add at beginning : ");
  scanf("%d",&ele);
  addatbeg(&head,ele);
  break;
  case 3:
  printf("\n Enter element and location : ");
  scanf("%d %d",&ele,&loc);
  addafter(head,ele,loc);
  break;
  case 4:
  display(head);
  break;
  case 5:
  printf ( "\nNo. of elements in the DLL = %d\n", count (
head ) ) ;
  break;
  case 6:
  printf("\n Enter element to delete : ");
  scanf("%d",&ele);
  delete(&head,ele);
  break;
  case 7:
  exit(0);
  break;
  default:
  printf("\n Invalid choice");
  }
  }
}

void append ( struct node **q, int num )
{
  struct node *r, *temp = *q ;

  if ( *q == NULL )
  {
    *q = malloc ( sizeof ( struct node ) ) ;
    ( *q ) -> prev = NULL ;
    ( *q ) -> data = num ;
    ( *q ) -> next = NULL ;
  }
  else
  {
    while ( temp -> next != NULL )
      temp = temp -> next ;

    r = malloc ( sizeof ( struct node ) ) ;
```

Empty list

As per user's choice operation will be performed

Creating first node

Module 3

```
        r -> data = num ;
        r -> next = NULL ;
        r -> prev = temp ;
        temp -> next = r ;
    }
}
```

temp is set to last node. New node *r* is created and its address is stored in temp->next

```
void addatbeg ( struct node **q, int num )
{
    struct node *temp ;
    temp = malloc ( sizeof ( struct node ) ) ;
    temp -> prev = NULL ;
    temp -> data = num ;
    temp -> next = *q ;
    ( *q ) -> prev = temp ;
    *q = temp ;
}
```

Address of previous first node *q* is stored in new node temp->next and *q* is set to temp to make it first node

```
void addafter ( struct node *q, int ele, int loc )
{
    struct node *temp ;
    int i ;
    for ( i = 0 ; i < loc ; i++ )
    {
        q = q -> next ;
        if ( q == NULL )
        {
            printf ( "\nInvalid location %d", loc );
            return ;
        }
    }
    q = q -> prev ;
    temp = malloc ( sizeof ( struct node ) ) ;
    temp -> data = ele ;
    temp -> prev = q ;
    temp -> next = q -> next ;
    temp -> next -> prev = temp ;
    q -> next = temp ;
}
```

Set q to node after which new node has to be added

Sets new node *temp*. *Temp*'s previous and next nodes are linked to *temp*.

```
void display ( struct node *q )
{
    while ( q != NULL )
    {
        printf ( " <-%d -> ", q -> data ) ;
        q = q -> next ;
    }
}
int count ( struct node * q )
{
    int cnt = 0 ;
    while ( q != NULL )
    {
        q = q -> next ;
        cnt++ ;
    }
    return cnt ;
}
void delete ( struct node **q, int num )
{
    struct node *temp = *q ;
    while ( temp != NULL )
    {
        if ( temp -> data == num )
        {
            if ( temp == *q )
            {
                *q = ( *q ) -> next ;
                ( *q ) -> prev = NULL ;
            }
            else
            {
                if ( temp -> next == NULL )
                    temp -> prev -> next = NULL ;
                else
                {
                    temp -> prev -> next = temp -> next ;
                    temp -> next -> prev = temp -> prev ;
                }
                free ( temp ) ;
```

if node to be deleted is the first node

if node to be deleted is the last node

if node to be deleted is intermediate node

```
        printf("\n Node deleted %d",num);
            }
            return ;
        }
        temp = temp -> next ;
    }
    printf ( "\n%d not found.", num ) ;
}
```

Output

```
C:\LINK4.exe

1 : Append
2 : Addatbeg
3 : Addafter
4 : Display
5 : Count
6 : Delete
7 : Exit
Select your choice : 1
```

3.9.3 Difference between Singly and Doubly and Circular Linked List

UQ. 3.9.22 *State differences between Singly Linked List and Doubly Linked list data structures along with their applications.* MU - May 15, Dec. 17, 5 Marks

Parameter	Singly Linked List	Doubly Linked List	Circular Linked List
Node Structure	Node contains two parts: data and link to next node	Node contains three parts: data and links to previous and next nodes	Node contains two parts: data and link to next node
Traversing	Only forward traversing is allowed	Forward and backward both traversing is allowed	Only forward traversing is allowed but can jump from last node to first
Memory	It uses less memory per node (single pointer)	It uses more memory per node(two pointers)	It uses less memory per node (single pointer)
Application	Singly linked list can mostly be used for stacks	Doubly linked list can be used to implement stacks, heaps, binary trees.	Can be used to implement round robin method
Complexity	Complexity of Insertion and Deletion at known position is O (n).	Complexity of Insertion and Deletion at known position is O (1).	At known, Position the Complexity of Insertion is O(n) an deletion is O(1)

Module 3

Syllabus Topic : Singly Linked List Applications

3.10 SINGLY LINKED LIST APPLICATIONS

UQ. 3.10.1 *Explain any one application of linked list with an example.* MU - May 16, 8 Marks

UQ. 310.2 *Explain different applications of linked-list.* MU - May 18, 5 Marks

- Linked Lists can be used to implement Stacks and Queues.
- Linked Lists can also be used to implement Graphs. (Adjacency list representation of Graph).
- Implementing Hash Tables : Each Bucket of the hash table can itself be a linked list. (Open chain hashing).
- Undo functionality in Photoshop or Word. Linked list of states.
- A polynomial can be represented in an array or in a linked list by simply storing the coefficient and exponent of each term.

- However, for any polynomial operation, such as addition or multiplication of polynomials, linked list representation is more easier to deal with.
- Linked lists are useful for dynamic memory allocation.
- The real life application where the circular linked list is used is our Personal Computers, where multiple applications are running.
- All the running applications are kept in a circular linked list and the OS gives a fixed time slot to all for running. The Operating System keeps on iterating over the linked list until all the applications are completed.

Syllabus Topic : Polynomial Representation and Addition

3.10.1 Polynomial Representation and Addition

UQ. 3.10.3 *Write short note on : Application of Linked-List-Polynomial addition.*

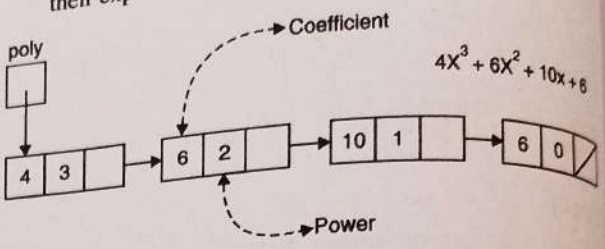
MU - May 19, 5 Marks

UQ. 3.10.4 *Explain following with suitable example : Polynomial representation and addition using linked list.* MU - Dec. 19, 10 Marks

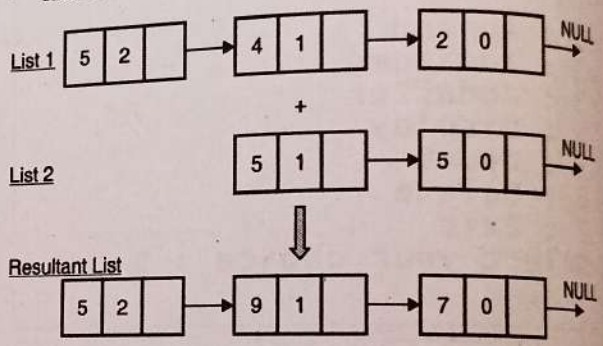
- A polynomial p(x) is the expression in variable x which is in the form ($ax^n + bx^{n-1} + \ldots + jx + k$), where a, b, c ...., k fall in the category of real numbers and 'n' is non negative integer, which is called the degree of polynomial.
- An important characteristics of polynomial is that each term in the polynomial expression consists of two parts:
  - one is the coefficient
  - other is the exponent

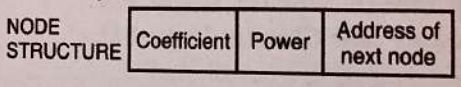
Example

- 10x2 + 26x, here 10 and 26 are coefficients and 2, 1 are its exponential value.
- Points to keep in Mind while working with Polynomials:
- The sign of each coefficient and exponent is stored within the coefficient and the exponent itself
- Additional terms having equal exponent is possible one
- The storage allocation for each term in the polynomial must be done in ascending and descending order of their exponent



Addition of two polynomials





Program on addition of polynomials

UQ. 3.10.5 *Write a C program to represent and add two polynomials using linked list.*

MU - Dec. 18, 12 Marks

```
// Node structure containing power and coefficient of variable
#include<stdio.h>
struct Node
{
  int coeff;
  int pow;
  struct Node *next;
};

void create_node(int x, int y, struct Node **temp)
```

Node structure containing power and coefficient of variable

```
{
  struct Node *r, *z;
  z = *temp;
  if(z == NULL)
  {
    r =(struct Node*)malloc(sizeof(struct Node));
    r->coeff = x;
    r->pow = y;
    *temp = r;
    r->next = (struct Node*)malloc(sizeof(struct Node));
    r = r->next;
    r->next = NULL;
  }
  else
  {
    r->coeff = x;
    r->pow = y;
    r->next = (struct Node*)malloc(sizeof(struct Node));
    r = r->next;
    r->next = NULL;
  }
}

void polyadd(struct Node *poly1, struct Node *poly2, struct
Node *poly)
{

while(poly1->next && poly2->next)
  {
    // If power of 1st polynomial is greater then 2nd, then
store 1st as it is
    // and move its pointer
    if(poly1->pow > poly2->pow)
    {
      poly->pow = poly1->pow;
      poly->coeff = poly1->coeff;
      poly1 = poly1->next;
    }
    else if(poly1->pow < poly2->pow)
    {
      poly->pow = poly2->pow;
      poly->coeff = poly2->coeff;
      poly2 = poly2->next;
    }

    // If power of both polynomial numbers is same then add
their coefficients
    else
    {
      poly->pow = poly1->pow;
      poly->coeff = poly1->coeff+poly2->coeff;
      poly1 = poly1->next;
      poly2 = poly2->next;
    }

    // Dynamically create new node
    poly->next = (struct Node *)malloc(sizeof(struct
Node));
    poly = poly->next;
    poly->next = NULL;
  }
while(poly1->next || poly2->next)
  {
    if(poly1->next)
    {
      poly->pow = poly1->pow;
      poly->coeff = poly1->coeff;
      poly1 = poly1->next;
    }
    if(poly2->next)
    {
      poly->pow = poly2->pow;
      poly->coeff = poly2->coeff;
      poly2 = poly2->next;
    }
    poly->next = (struct Node *)malloc(sizeof(struct
Node));
    poly = poly->next;
```

Function to create new node

Function for adding two polynomials

If power of 2nd polynomial is greater then 1st, then store 2nd as it is and move its pointer

Module 3

```
        poly->next = NULL;
    }
}

// Display Linked list
void show(struct Node *node)
{
while(node->next != NULL)
    {
    printf("%dx ^ %d", node->coeff, node->pow);
    node = node->next;
    if(node->next != NULL)
        printf(" + ");
    }
}

// Driver program
int main()
{
    struct Node *poly1 = NULL, *poly2 = NULL, *poly =
NULL;

    // Create first list of 5x^2 + 4x^1 + 2x^0
    create_node(5,2,&poly1);
    create_node(4,1,&poly1);
    create_node(2,0,&poly1);

    // Create second list of 5x^1 + 5x^0
    create_node(5,1,&poly2);
    create_node(5,0,&poly2);

    printf("1st Number: ");
    show(poly1);

    printf("\n2nd Number: ");
    show(poly2);

    poly = (struct Node *)malloc(sizeof(struct Node));

    // Function add two polynomial numbers
    polyadd(poly1, poly2, poly);

    // Display resultant List
    printf("\nAdded polynomial: ");
    show(poly);

getch();
}
```

Output

```
C:\poly.exe
1st Number: 5x^2 + 4x^1 + 2x^0
2nd Number: 5x^1 + 5x^0
Added polynomial: 5x^2 + 9x^1 + 7x^0
```

..Chapter End