**UNIT –II -CLO-1**

**Part-A**

**Multiple Choice Questions**

**1. An array is a collection of \_\_\_\_\_\_\_\_\_\_ data elements.**

a.) Simple

b) Complex

c) Similar

d) Dissimilar

**Ans: c**

**2. \_\_\_\_\_\_\_\_\_\_ an array means accessing each and every element of the array for a specific purpose. [Pg. No.71]**

a) Traversing

b) Inserting

c) Searching

d) Sorting

**Ans: a**

**3. Merging the arrays is very simple if arrays are \_\_\_\_\_\_\_\_\_ [Pg.No.82]**

a) Sorted

b) Unsorted

c)equal

d)not equal

**Ans: b**

**4.\_\_\_\_\_\_\_\_\_\_\_ is a matrix that has large number of elements with a zero value. [Pg.No.110]**

a) Sparse matrix

b) dense matrix

c)Simple matrix

d)Complex matrix

**Ans: a**

**5.** **Linked List does not allow \_\_\_\_ access of data [Pg.No.164]**

a) Sequential

b) random

c) Both sequential and random

d) None of these

**Ans: b**

**6. A\_\_\_\_\_\_\_\_ linked list allows traversal of data only in one way. [Pg.no.167]**

a) Singly

b) doubly

c)circular

d)None

**Ans: a**

*// Not Circular as it can be singly circular list or doubly circular list*

**7.\_\_\_\_\_\_\_\_\_ linked lists are widely used in operating systems for task maintenance. [Pg.no.180]**

a) Singly

b) doubly

c)circular

d)None

**Ans: c**

**8. The main advantage of using a \_\_\_\_\_\_\_ linked list is that it makes searching twice as efficient. [Pg.no.188]**

a) Singly

b) doubly

c)circular

d)None

**Ans: b**

**9.\_\_\_\_\_\_\_\_ stores the address of the first node in the list.**

a) START

b) END

c) NEXT

d) NULL

**Ans: a**

**10.\_\_\_\_\_\_\_\_\_ is a condition that occurs when we try to delete a node from linked list that is empty.**

a) Overflow

b) Underflow

c)NULL

d)None

**Ans: b**

**11.In a circular linked list**a) Components are all linked together in some sequential manner.  
b) There is no beginning and no end.  
c) Components are arranged hierarchically.  
d) Forward and backward traversal within the list is permitted.

**Ans: b**

**12. Which of the following operations is performed more efficiently by doubly linked list than by singly linked list?**  
a) Deleting a node whose location in given  
b) Searching of an unsorted list for a given item  
c) Inserting a node after the node with given location  
d) Traversing a list to process each node

**Ans:** a

**13.In linked list each node contains minimum of two fields. One field is data field to store the data second field is?**  
a) Pointer to character  
b) Pointer to integer  
c) Pointer to node  
d) Node

**Ans: c**

**14.A variant of linked list in which last node of the list points to the first node of the list is?**  
a) Singly linked list  
b) Doubly linked list  
c) Circular linked list  
d) Multiply linked list

**Ans:c**

**15. In doubly linked lists, traversal can be performed?**  
a) Only in forward direction  
b) Only in reverse direction  
c) In both directions  
d) None

**Ans:c**

**16.What kind of linked list is best to answer question like “What is the item at position n?”**  
a) Singly linked list  
b) Doubly linked list  
c) Circular linked list  
d) Array implementation of linked list

**Ans: d**

**17.A variation of linked list is circular linked list, in which the last node in the list points to first node of the list. One problem with this type of list is?**  
a) It waste memory space since the pointer head already points to the first node and thus the list node does not need to point to the first node.  
b) It is not possible to add a node at the end of the list.  
c) It is difficult to traverse the list as the pointer of the last node is now not NULL  
d) All of above

**Ans: c**

**18.A variant of the linked list in which none of the node contains NULL pointer is?**  
a) Singly linked list  
b) Doubly linked list  
c) Circular linked list  
d) None

**Ans: a**

**20. In circular linked list, insertion of node requires modification of?**  
a) One pointer  
b) Two pointer  
c) Three pointer  
d) None

**Ans :b**

**PART-B**

**Q1. Write an algorithm to traverse a linked list**

1. Set PTR = START

2. Repeat steps 3 and 4 while PTR is not equal to NULL.

3. Apply PROCESS to INFO [PTR]

4. Set PTR: = LINK [PTR]

5. Exit

**Q2. What are the limitations of arrays?**

The following are the limitations of arrays:

1. Arrays are of fixed size.
2. Data elements are stored in continuous memory locations which may not be available always.

3) Adding and removing of elements is problematic because of shifting the locations.

**Q3. What is the difference between an array and a linked list?**

1. The size of an array is fixed whereas size of linked list is variable.
2. In array the data elements are stored in continuous memory locations but in linked listit is non continuous memory locations.
3. Addition, removal of data is easy in linked list whereas in arrays it is complicated.

**Q4. Write the syntax of node creation?**

**Syntax:**

struct node

{

data type ;

struct node \*ptr; **//pointer for link node**

}

temp;

**Q5. Explain traversing a linked list.**

* We have already learnt that linked list is a collection of elements called node.
* Each node has two components:

1. The first component contains the data (or information) of the element and
2. the second component called the next field (or linked field) which holds the address of the next node.

* To indicate the end of the list, we generally assign a NULL value.
* A node in a single linked list has the following structure:

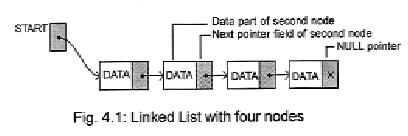
struct node

{  
int data;

Struct node \*next;

};

Here member of the structure struct node \*link points to the structure itself.

****

**Traversing a linked list :**

* Traversing a linked list means to visit every node or element in the list starting from first to last.
* For traversing a linked list we consider ptr as a pointer varible.
* The linked list also contains a pointer variable called start which contains the address of the first node in the list.
* NULL pointer denoted by x in the figure indicated the end of the list.
* Initially we assign the value of start to ptr.
* Thus ptr will points to the first element of the list. For processing the next element we assign the address of the next element to the ptr as

ptr=ptr->next;

* Now ptr holds the address of the next node of the list.
* We can traverse each element of the linked list through this assignment untillptr has NULL address (which denotes the last node of the list).
* Thus the traversal of a linked list can be implemented with the following statements
* while(ptr!=NULL)  
  ptr = ptr -> next;

**Q6.Explain Inserting a linked list.**

* There may be two possibilities for inserting a node depending upon the position of insertion.
* The two ways are

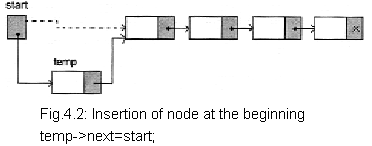
1. Insertion at the beginning
2. Insertion in between

**Insertion at the beginning**

* Let us consider a pointer temp which points to the node that has to be inserted.
* The data part of the new node is assigned to the data (i.e., d) through the temp pointer as :

temp->data=d;

* Again we know that the start pointer points to the first node of the linked list.
* Therefore, for inserting at the beginning we will have to assign start to the next part of temp i.e.,



* Now inserted node points to the next node which was the first node of the linked list.
* So the inserted node is the first node of the linked list and start pointer is reassigned as :

start = temp;

**Insertion in between**

* For inserting a node at any position of the linked list (except at the beginning) we will have to first traverse the linked list for obtaining the node after which we want to insert the new node.
* Let us consider a pointer temp which points to the node that has to be inserted. The data part of the new node is assigned to the data (i.e., d) through the temp pointer as :

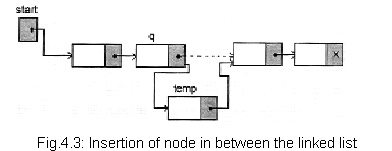
**temp->data=d;**

* Again we consider a pointer q which points to the node after which we have to insert the new node.
* For inserting the element after the node, we assign the next part of the node to the next part of new inserted node as:

**temp->next=q->next;**

* Then the address part of the new inserted node is assigned to the next part of the previous node.

q->next=temp;



**Q7.Explain deletion from a linked list.**

* An element that is to be deleted from a linked list is to be searched first.
* For this, the traversing operation must be carried out thoroughly on the list.
* After finding the element there may be two cases for deletion:  
  1.Deletion at beginning  
  2.Deletion in between

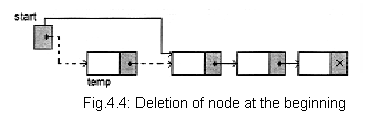
**Deletion at beginning** :

* We know that start pointer points to the first element of a linked list.
* If element to be deleted is the first element of linked list then we assign the value of start to temp as:

temp=start ;

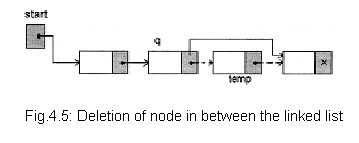
* So now temp points to first node which has to be deleted. Now we assign the next part of the deleted node to start as:

start=start->next;

* Since start points to the first element of the linked list, so start->next will point to the second element of the linked list.
* Now we should free the element to be deleted which is pointed by temp with the following statement:  
  free(temp);   
  

**Deletion in between :**

* If the element is other than the first element of the linked list then we assign the next part of the deleted node to the next part of the previous node.
* This can be written as:  
  temp=q->next;  
  q->next=temp->next;  
  free(temp);
* Here, the pointer q is pointing to the previous node of the node to be deleted.
* After the first statement temp will point to the node to be deleted, after second statement next of previous node will point to next node of the node to be deleted.
* If the node to be deleted is the last node of the linked list then the second statement will be as:

q->next=NULL;  
  


**Q8.Ellaborate on josephus problem.**

The **Josephus problem** (or **Josephus permutation**) is a theoretical problem related to a certain [counting-out game](https://en.wikipedia.org/wiki/Counting-out_game). People are standing in a [circle](https://en.wikipedia.org/wiki/Circle) waiting to be executed. Counting begins at a specified point in the circle and proceeds around the circle in a specified direction. After a specified number of people are skipped, the next person is executed. The procedure is repeated with the remaining people, starting with the next person, going in the same direction and skipping the same number of people, until only one person remains, and is freed. The problem — given the number of people, starting point, direction, and number to be skipped — is to choose the position in the initial circle to avoid execution.

int survivor(struct node \*\*head,int k)

{

struct node \*p,\*q;

int i;

q = p =\*head;

while(p->next != p)

{

for(i =0; i < k -1; i++)

{

q = p;

p = p->next;

}

q->next = p->next;

printf("%d has been killed.\n", p->num);

free(p);

p = q->next;

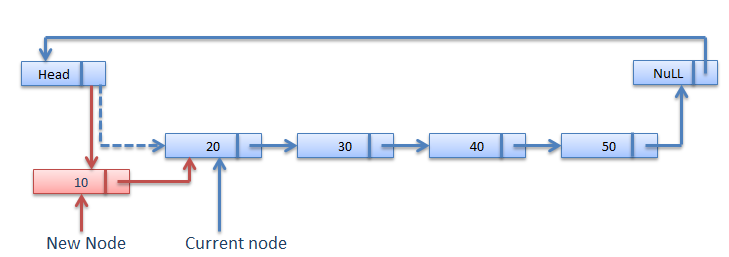
}

\*head = p;

return(p->num);}

**Q9. Explain insertion of a node in front in a circular linked list.**

**Procedure for insertion a node at the beginning of list**

Step1. Create the new node  
Step2. Set the new node’s next to itself (circular!)   
Step3. If the list is empty,return new node.  
Step4. Set our new node’s next to the front.  
Step5. Set tail’s next to our new node.   
Step6. Return the end of the list.  
  


void insert\_first(struct link \*node){

node=start->next;

ptr=start;

new1=(struct link \*)malloc(sizeof(struct link));

printf("\n Insert data for first node:");

scanf("%d",&new1->data);

ptr->next=new1;

new1->next=node;

i++;

}

**Q10. Explain insertion at middle in circular linked list.**

voidinsert\_location(struct link \*node)

{

intnode\_no=1,insert\_no,flag=0,count;

node=start->next;

ptr=start;

count=i;

printf("\n Enter position where you want to insert new node:-");

scanf("%d",&insert\_no);

while(count)

{

if(node\_no==insert\_no)

{

new1=(struct link \*)malloc(sizeof(struct link));

printf("\n Insert data for new node:-");

scanf("%d",&new1->data);

ptr->next=new1;

new1->next=node;

flag=1;

break;

}

else

{

ptr=ptr->next;

node=node->next;

}

node\_no++;

count--;

}

if(flag==0)

{

printf("\n Position not found");

}

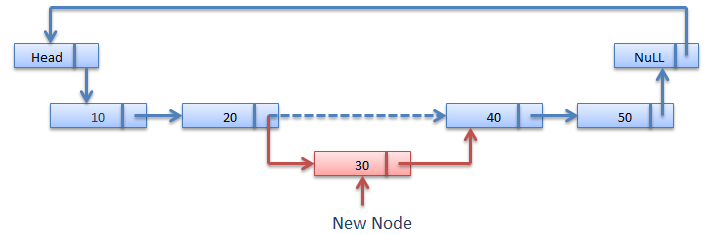
else

{

i++;

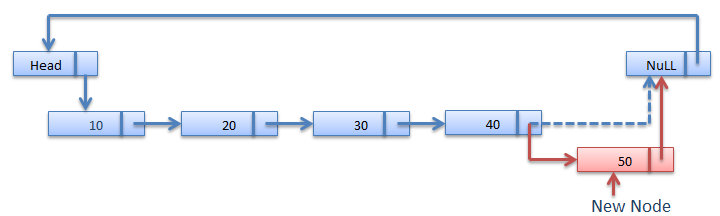
}

}



**Q11.Explain insertion at end in circular linked list**

**Procedure for insertion a node at the Last of list**

Step1. Create the new node  
Step2. Set the new node’s next to itself (circular!)   
Step3. If the list is empty,return new node.  
Step4. Set our new node’s next to the front.  
Step5. Set tail’s next to our new node.   
Step6. Return the end of the list.  
  


**Algorithm for Insertion at the last of Circular linked list**

node\* AddEnd(node\* tail, intnum)

{

node \*temp = (node\*)malloc(sizeof(node));

temp->data = num;

temp->next = temp;

if (tail == NULL)

return temp;

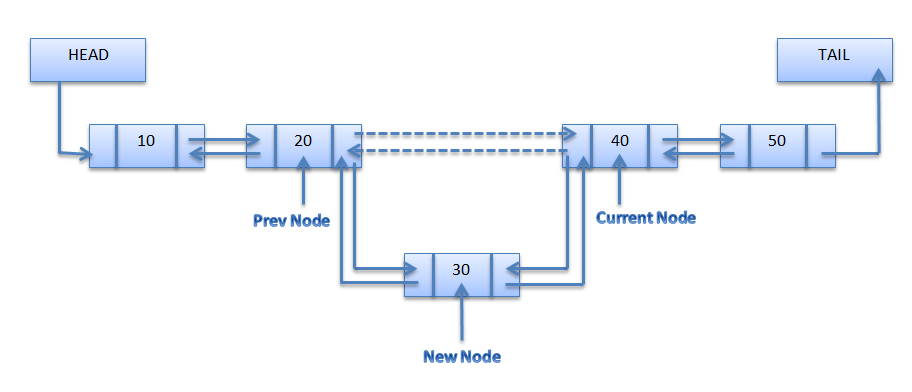
temp->next = tail->next;

tail->next = temp;

return temp;

}

**Q12.Explain Insertion at middle in doubly linked list.**



**FuctionFor Insert at Location**

voidinsertAtloc(node \*\*start,node \*\*end,int item , inti,int size )

 {

 node \*ptr,\*loc;

 int n=1 ;

 i=i-1;

 ptr=(node\*)malloc(sizeof(node));

 ptr->info=item;

 loc = \*start ;

if(\*start==NULL)

 {

  ptr->prev = ptr->next = NULL ;

  \*start = \*end = ptr ;

 }

 else if(i<=size)

 { while(n != i)

  {

loc=loc->next;

n++;

}

  ptr->next = loc->next ;

  ptr->prev = loc ;

(loc->next)->prev = ptr ;

  loc->next = ptr ;}

 else

 {

ptr->prev = \*end;

  (\*end)->next = ptr ;

  ptr->next= NULL;

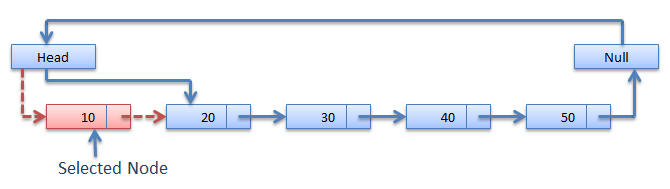
  (\*end)=ptr;

  }

 }

**Q13. Explain deletion at front in circular linked list.**

#### Deletion at beginning of the Circular linked list



void delete\_first(struct link \*node)

{

node=start->next;

ptr=start;

if(i==0)

{

printf("\n List is empty");

exit(0);

}

ptr->next=node->next;

free(node);

i--;

}

**Q14.Explain deletion at the end(at last node) In circular linked list.**

void delete\_last(struct link \*node)

{

  intnode\_no=0,count;

  node=start->next;

  ptr=start;

  count=i;

  if(i==0)

  {

  printf("\n List is empty");

  exit(0);

}

  while(count)

  {

  node\_no++;

  ptr=ptr->next;

  node=node->next;

  count--;

  }

node=start->next;

  ptr=start;

  while(node\_no!=1)

  {

  node\_no--;

  ptr=ptr->next;

  node=node->next;

  }

  if(node\_no==1)

  {

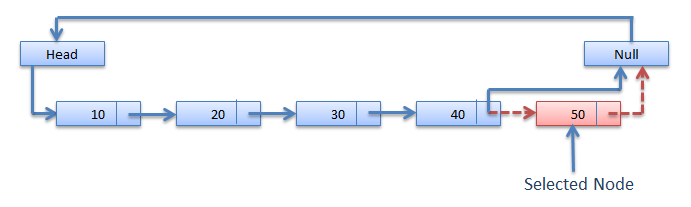
  ptr->next=node->next;

  free(node);

  i--;

  }

 }



**Q15.Explain searching for a node in doubly linked list.**

The doubly linked list can be traversed in any order to reach the given element. The following listing demonstrates how to search an element from the beginning.

 node \*search (node \*head, int item)

{

while(head!=NULL)

{

if(head->info==item) // if the values match,

return head; // return the matching node.

head=head->fnext; // otherwise, move on

}

return NULL;

}

**Q16 Differentiate between circular linked list and doubly linked list.**

In the last [node](https://en.wikipedia.org/wiki/Node_%28computer_science%29) of a list, the link field often contains a [null](https://en.wikipedia.org/wiki/Null_pointer#Null_pointer) reference, a special value used to indicate the lack of further nodes. A less common convention is to make it point to the first node of the list; in that case the list is said to be 'circular' or 'circularly linked'; otherwise it is said to be 'open' or 'linear'.

[Circularly-linked-list.svg](https://en.wikipedia.org/wiki/File:Circularly-linked-list.svg)  
*A circular linked list*

In the case of a circular doubly linked list, the only change that occurs is that the end, or "tail", of the said list is linked back to the front, or "head", of the list and vice versa.

### Doubly linked list

In a 'doubly linked list', each node contains, besides the next-node link, a second link field pointing to the 'previous' node in the sequence. The two links may be called 'forward('s') and 'backwards', or 'next' and 'prev'('previous').

[Doubly-linked-list.svg](https://en.wikipedia.org/wiki/File:Doubly-linked-list.svg)

A doubly linked list whose nodes contain three fields: an integer value, the link forward to the next node, and the link backward to the previous node

**PART-C**

1. Explain array and its operations in detail.
2. Why does storing of sparse matrices need extra consideration? How are sparse matrices stored efficiently in computer’s memory?
3. Write a program to interchange the second elements with the second last element.
4. Write a program to create a linked list and perform insertions and deletionsof all cases.
5. i) Give the linked representation of the following polynomial

**7x^3 y^2 – 8 x^2 y + 3xy+11x-4.**

ii)Make a comparision between a linked list and a linear array. Which one will you prefer to use and when?

1. Explain insertion and deletion operations in a singly linked list.
2. Explain all three insertions in circular linked list.
3. Explain deletion in Circular linked list.
4. What is circular linked list.Give an application and explain it.
5. Explain all three insertions in Doubly linked list.
6. Explain all three insertions in circular linked list.
7. Explain deletion in Circular linked list.
8. What is circular linked list? Give an application and explain it.
9. Explain all three insertions in doubly linked list.
10. Explain all three deletions in doubly linked list.
11. Explain Insertion and Searching in Doubly linked list.