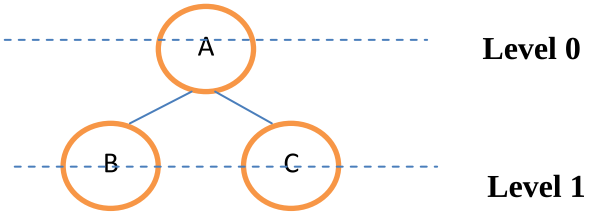
WINTER– 18 EXAMINATION

Subject Name: Data Structure using C **Model Answer** Subject Code: 22317

Important Instructions to examiners:

1) The answers should be examined by key words and not as word-to-word as given in the model answer scheme.
2) The model answer and the answer written by candidate may vary but the examiner may try to assess the understanding level of the candidate.
3) The language errors such as grammatical, spelling errors should not be given more Importance (Not applicable for subject English and Communication Skills.
4) While assessing figures, examiner may give credit for principal components indicated in the figure. The figures drawn by candidate and model answer may vary. The examiner may give credit for any equivalent figure drawn.
5) Credits may be given step wise for numerical problems. In some cases, the assumed constant values may vary and there may be some difference in the candidate's answers and model answer.
6) In case of some questions credit may be given by judgement on part of examiner of relevant answer based on candidate's understanding.
7) For programming language papers, credit may be given to any other program based on equivalent concept.

Q. No.	Sub Q. N.	Answer	Marking Scheme
1		**Attempt any FIVE of the following :**	**10 M**
	a	**Define the term algorithm.**	2 M
	Ans	Algorithm is a stepwise set of instructions written to perform a specific task.	Correct definition 2M
	b	**List any 4 applications of queue.**	2 M
	Ans	• In computer system to maintain waiting list for single shared resources such as printer, disk, etc. • It is used as buffers on MP3 players, iPod playlist, etc. • Used for CPU scheduling in multiprogramming and time sharing systems. • In real life, Call Center phone systems will use Queues, to hold people calling them in an order, until a service representative is free. • Handling of interrupts in real-time systems. • Simulation	Any four apllications-1/2 M each
	c	**Describe following terms w.r.to tree:** **(i) Leaf node** **(ii) Level of node**	2 M
	Ans	**Example:**	Description of each term 1M



(i) Leaf node: A node without any child node is called as leaf node.

Nodes B and C are leaf node as shown in above example.

(ii) Level of node: Position of a node in the hierarchy of a tree is called as level of node.

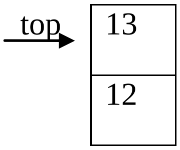
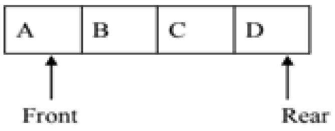
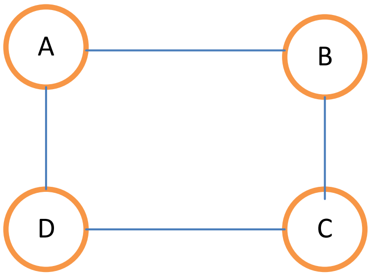
Level of node B is 1 as shown in above example.

d **Differentiate between stack and queue.(Any two points)** 2 M

Ans

Stack	**Queue**
1. Stack is a data structure in which insertion and deletion operations are performed at **same end**.	1. Queue is a data structure in which insertion and deletion operations are performed at **different ends**.
2. In stack an element inserted last is deleted first so it is called **Last In First Out list.**	2. In Queue an element inserted first is deleted first so it is called **First In First Out list.**
3.In stack only one pointer is used called as **stack top**	3.In Queue two pointers are used called as **front** and **rear**
4. **Example**: Stack of books	4. **Example**: Students standing in a line at fees counter
5.**Application**: • Recursion • Polish notation	5. **Application**: • In computer system for organizing processes. • In mobile device for sending receiving messages.

Any two correct differences-1M each

<table>
<tr><td></td><td></td><td><table><tr><td><b>6. Representation</b>: Using array
 </td><td><b>6. Representation:</b> Using array
 </td></tr></table></td><td></td></tr>
<tr><td></td><td><b>e</b></td><td><b>Describe undirected graph with suitable example.</b></td><td>2 M</td></tr>
<tr><td></td><td><b>Ans</b></td><td>Undirected graph: A graph in which the edges do not have any direction associated with them is known as undirected graph.
In undirected graph, if an edge exists between two nodes A and B then the nodes can traverse from A to B as well as from B to A. Each edge is bidirectional.
Example:-

In the above example, each edge is bidirectional.</td><td>Definition-1M, example-1M</td></tr>
<tr><td></td><td><b>f</b></td><td><b>Define the terms: Linear data structure and non-linear data structure.</b></td><td>2 M</td></tr>
<tr><td></td><td><b>Ans</b></td><td>Linear Data Structure: A data structure in which all data elements are stored in a particular sequence is known as linear data structure.
Example: stack, queue
Non-Linear data structure: A data structure in which all data elements are not stored in any particular sequence is known as nonlinear data structure.
Example: graph and tree.</td><td>Each term definition 1M</td></tr>
<tr><td></td><td><b>g</b></td><td><b>convert infix expression into prefix expression:</b>
<b>(A+B)*(C/G)+F</b></td><td>2 M</td></tr>
<tr><td></td><td><b>Ans</b></td><td><table><tr><th>Infix expression</th><th>Read Character</th><th>Stack contents</th><th>Prefix expression</th></tr><tr><td><b>(A+B)*(C/G)+F</b></td><td><b>F</b></td><td><b>-</b></td><td><b>F</b></td></tr><tr><td><b>(A+B)*(C/G)+</b></td><td>+</td><td>+</td><td><b>F</b></td></tr></table></td><td>Correct prefix expression - 2M</td></tr>
</table>

(A+B)*(C/G)	**)**	**+)**	**F**
(A+B)*(C/G	**G**	**+)**	**GF**
(A+B)*(C/	**/**	**+)/**	**GF**
(A+B)*(C	**C**	**+)/**	**CGF**
(A+B)*(	**(**	**+**	**/CGF**
(A+B)*	*****	**+***	**/CGF**
(A+B)	**)**	**+*)**	**/CGF**
(A+B	**B**	**+*)**	**B/CGF**
(A+	**+**	**+*)+**	**B/CGF**
(A	**A**	**+*)+**	**AB/CGF**
(	**(**	**+***	**+AB/CGF**
			***+AB/CGF**
			+*+AB/CGF

2		**Attempt any THREE of the following :**	**12 M**
	a	**Describe working of linear search with example.**	4 M
	Ans	In linear search, search element is compared with each element from the list in a sequence. Comparison starts with first element from the list and continues till number is found or comparison reaches to the last element of the list.	Relevant description 2M, Any correct example-2M

As each element is checked with search element, the process of searching requires more time. Time complexity of linear search is O (n) where n indicates number of elements in list.

Linear search on sorted array:-On sorted array search takes place till element is found or comparison reaches to an element greater than search element.

Example:- Using array representation

Input list 10, 20, 30, 40, 50 and Search element 30, Index =0

Iteration 1

10	20	30	40	50

10 ! = 30

Index = Index + 1

Iteration 2

10	20	30	40	50

↑

20 ! = 30

Index = Index + 1

Iteration 3

10	20	30	40	50

↑

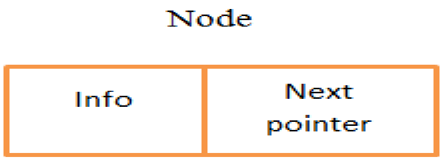
30 = 30

Number found

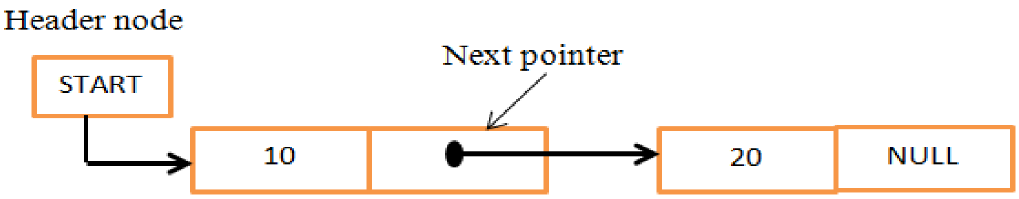
b **Describe the concept of linked list with the terminologies: node, next Pointer, null pointer and empty list.** 4 M

Ans **Node**: Each data element in a linked list is represented as a node. Node contains two parts- one is info (data) and other is next pointer (address). Info part stores data and next pointer stores address of next node.

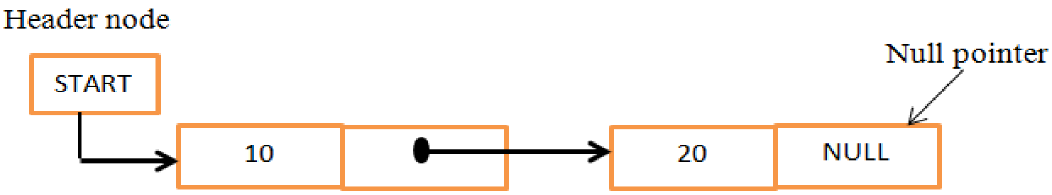
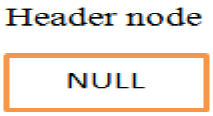
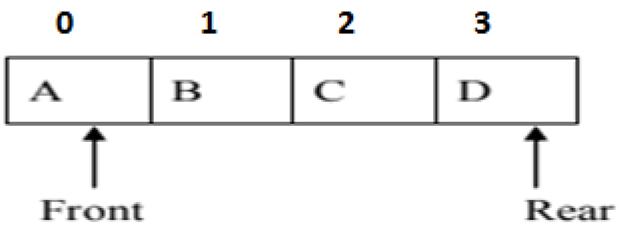
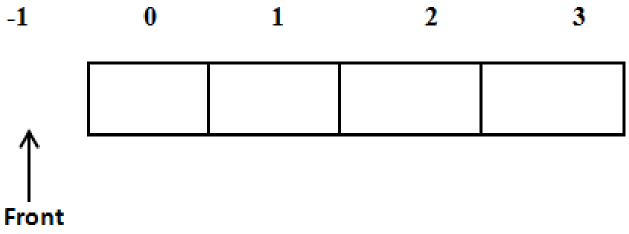
Description of each terminology -1M



Next pointer: It is a pointer that holds address of next node in the list i.e. next pointer points to next node in the list

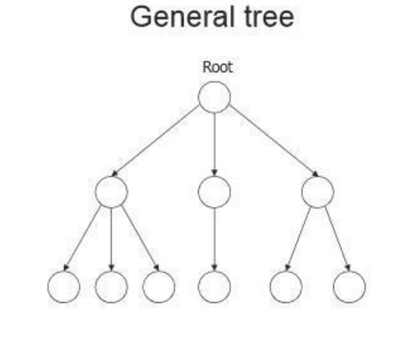
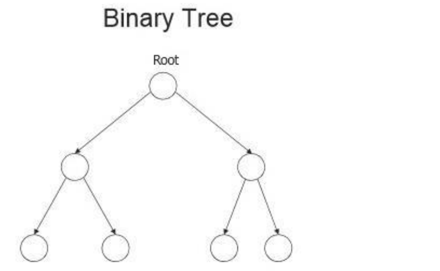


Null pointer: It is a pointer that does not hold any memory address i.e. it is pointing to nothing. It is used to specify end of the list. The last element of list contains NULL pointer to specify end of list.

		 **Empty list**: Each linked list has a header node. When header node contains NULL value, then that list is said to be empty list. 	
	c	**Describe queue full and queue empty operation conditions on linear queue with suitable diagrams.**	4 M
	Ans	**Queue full**:-A queue is full when its rear pointer points to max -1 position. Max is maximum number of elements in a queue. If rear pointer is not equal to max-1 then a new element can be added to a queue. If queue is full then new element cannot be added to a queue. Example:- Consider max=4. First element is stored at 0th position and last element is stored at 3rd position in queue. In the diagram given below rear pointer is pointing to max-1 (3) position so queue is full.  **Queue empty**: A queue is empty when its front pointer points to -1 position. When front pointer is -1 then one cannot delete any data from a queue. Example:-In the diagram given below front pointer points to -1 value i.e. it points no location inside queue so queue is empty. 	Description of queue full-1M, diagram-1M, Description of queue empty-1M, diagram-1M

	d	**Differentiate between general tree and binary tree. (any four points)**	4 M

Ans

Sr. no	General Tree	Binary Tree
1	A general tree is a **data structure** in which each node can have infinite number of children	A Binary tree is a data structure in which each node has at most **two nodes** i.e**.** left and right
2	In general tree, root has **in-degree 0** and maximum **out-degree n**.	In binary tree, root has **in-degree 0** and maximum **out-degree 2**.
3	In general tree, each **node** have in-degree **one** and maximum out-degree **n**.	In binary tree, each node have in-degree **one** and maximum out-degree **2**.
4	**Height** of a general tree is the length of longest path from root to the leaf of tree. Height(T) = {**max**(height(child1) , height(child2) , … height(child-n)) +1}	Height of a binary tree is : Height(T) = { **max** (Height(Left Child) , Height(Right Child) + 1}
5	Subtree of general tree are **not ordered**	Subtree of binary tree is **ordered**.
6	 General tree	 Binary Tree

Any four relevant differences -1M each

3		**Attempt any THREE of the following :**	**12 M**
	a	**Write a C program for deletion of an element from an array.**	4 M

Ans

```
#include <stdio.h>
int main()
{
  int array[100], position, c, n;
  printf("Enter number of elements in array\n");
  scanf("%d", &n);
  printf("Enter %d elements\n", n);
  for (c = 0; c < n; c++)
    scanf("%d", &array[c]);
  printf("Enter the location where you wish to delete element\n");
  scanf("%d", &position);
  if (position >= n+1)
```

4M for correct logic & program code

```
        printf("Deletion not possible.\n");
      else
      {
        for (c = position - 1; c < n - 1; c++)
          array[c] = array[c+1];

        printf("Resultant array:\n");

        for (c = 0; c < n - 1; c++)
          printf("%d\n", array[c]);
      }
       return 0;
    }
```

b **Convert following expression into postfix form. Give stepwise procedure.**

A+B↑C*(D/E)-F/G.

4 M

Ans Consider input expression as (A+B↑C*(D/E)-F/G)

Scanned Symbol	Operation stack	Postfix Expression
(	(	
A	(	A
+	(+	A
B	(+	AB
↑	(+↑	AB
C	(+↑	ABC
*	(+*	ABC↑
(	(+*(	ABC↑
D	(+*(	ABC↑D
/	(+*(/	ABC↑D
E	(+*(/	ABC↑DE
)	(+*	ABC↑DE/
-	(-	ABC↑DE/*+
F	(-	ABC↑DE/*+F

Correct Postfix Expression 4M

/	(-/	ABC↑DE/*+F
G	(-/	ABC↑DE/*+FG
)	EMPTY	ABC↑DE/*+FG/-

POSTFIX EXPRESSION: ABC↑DE/*+FG/-

c **Find the position of element 29 using binary search method in an array 'A' given below. Show each step.**

A={11,5,21,3,29,17,2,43}

4 M

Ans An array which is given A[ ]= {11,5,21,3,29,17,2,43} is not in sorted manner, first we need to sort them in order;

1M for taking sorted input & 1M each for every iteration

So an array will be A[ ]={2,3,5,11,17,21,29,43} and the value to be searched is VAL = 29.

The binary search algorithm will proceed in the following manner.

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]
2	3	5	11	17	21	29	43

Iteration 1:

BEG = 0, END = 7, MID = (0 + 7)/2 = 3

Now, VAL = 29 and A[MID] = A[3] =11

A[3] is less than VAL, therefore, we now search for the value in the second half of the array.

So, we change the values of BEG and MID.

Iteration 2:

Now, BEG = MID + 1 = 4, END = 7, MID = (4 + 7)/2 =11/2 = 5; VAL = 29 and A [MID] = A [5] = 21

A[5] is less than VAL, therefore, we now search for the value in the second half of the segment.

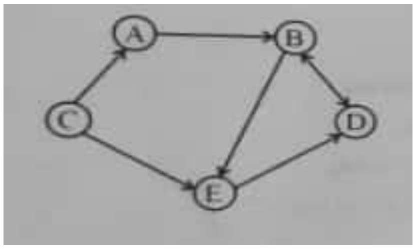
So, again we change the values of BEG and MID.

Iteration 3:

Now, BEG = MID + 1 = 6, END = 7, MID = (6 + 7)/2 = 6 Now, VAL = 29 and A [MID] = A [6]=29

So, Element 29 is found at 6th location in given array A[]={2,3,5,11,17,21,29,43}.

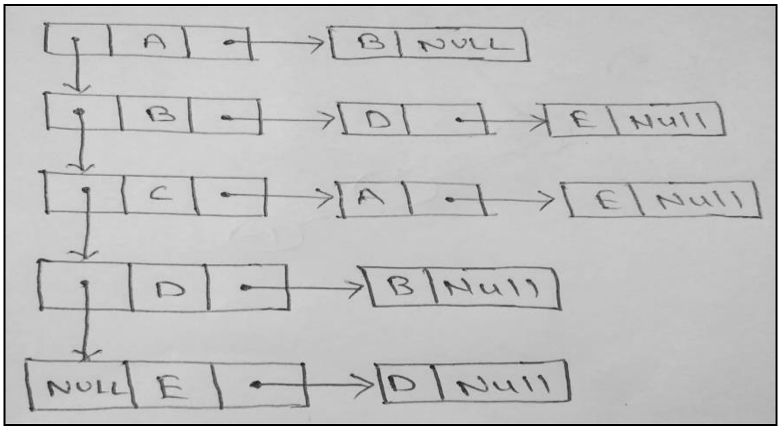
d give adjacency list and adjacency matrix for given graph: 4 M



Ans Adjacency List: (Using Linked List)

2M for Correct List and 2M for Correct matrix

Here, we use doubly linked list for storing header node list and singly linked list for storing respective adjacent node to it.



OR

Adjacency List

Nodes	**Adjacent Nodes**
A	**B**
B	**D,E**
C	**A,E**
D	**B**
E	**D**

Adjacency Matrix: (Using Array)

$$\begin{array}{c|ccccc|}
A & 0 & 1 & 0 & 0 & 0 \\
B & 0 & 0 & 0 & 1 & 1 \\
C & 1 & 0 & 0 & 0 & 1 \\
D & 0 & 1 & 0 & 0 & 0 \\
E & 0 & 0 & 0 & 1 & 0
\end{array}$$

4		**Attempt any THREE of the following :**	**12 M**
	a	**Describe working of bubble sort with example.**	4 M
	Ans		2M for description & 2M for example

Bubble sort is a simple sorting algorithm. This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order. This algorithm is not suitable for large data sets as its average and worst case complexity is of O (n^2) where **n** is the number of items.

Bubble Sort Working:

We take an unsorted array for our example as A[]={19, 2, 27, 3, 7, 5, 31}. Bubble sort takes O(n^2) time so we're keeping it short and precise.

{{**Note: Pass 4 onwards optional**}}

Pass 1: 2,19,27,3,7,5,31

2,19,27,3,7,5,31

2,19,3,27,7,5,31

2,19,3,7,27,5,31

2,19,3,7,5,27,31

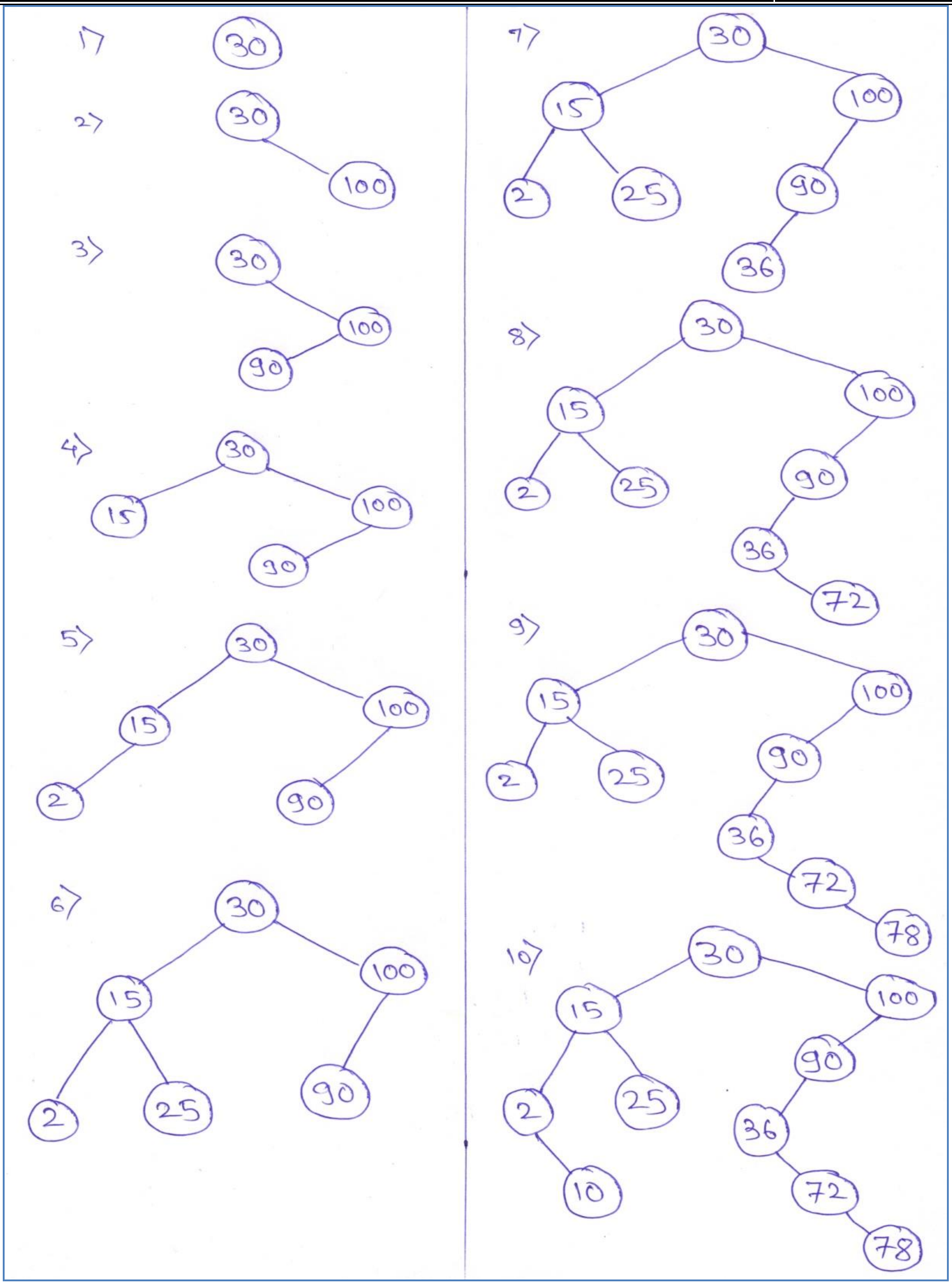
Pass 1 Completed

Pass 2: 2,19,3,7,5,27,31

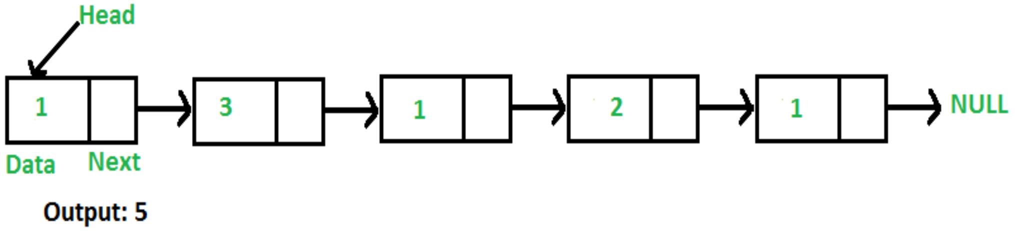
2,3,19,7,5,27,31

2,3,7,19,5,27,31

		2,3,7,5,19,27,31 2,3,7,5,19,27,31 Pass 2 Completed Pass 3: 2,3,7,5,19,27,31 2,3,7,5,19,27,31 2,3,5,7,19,27,31 Pass 3 Completed Pass 4: 2,3,5,7,19,27,31 Pass 4 Completed Pass 5: 2,3,5,7,19,27,31 Pass 5 Completed Pass 6: 2,3,5,7,19,27,31 Pass 6 Completed	
	b	**Construct a binary search tree for following elements:** **30,100,90,15,2,25,36,72,78,10 show each step of construction of BST.**	4 M
	Ans	Stepwise construction of Binary search tree for following elements: 30,100,90,15,2,25,36,72,78,10 is as follows:	4M for all correct steps

<table>
<tr><td></td><td></td><td>

1)
30
2)
30
100
3)
30
100
90
4)
30
15
100
90
5)
30
15
100
2
90
6)
30
15
100
2
25
90
7)
30
15
100
2
25
90
36
8)
30
15
100
2
25
90
36
72
9)
30
15
100
2
25
90
36
72
78
10)
30
15
100
90
2
25
36
10
72
78

</td><td></td></tr>
<tr><td></td><td>c</td><td><b>Write an algorithm to count number of nodes in singly linked list.</b></td><td>4 M</td></tr>
<tr><td></td><td><b>Ans</b></td><td>Function to count number of nodes in a given singly linked list.


Head
1
3
1
2
1
NULL
Data
Next
Output: 5

</td><td>4M for correct algorithm</td></tr>
</table>

For example, the function should return 5 for linked list 1->3->1->2->1.

Algorithm: Using Iterative Solution

```
1) Initialize count as 0
2) Initialize a node pointer, current = head.
3) Do following while current is not NULL
    a) current = current -> next
    b) count++;
4) Return count
```

d **Write a program in ‘C’ to insert an element in a linear queue.** 4 M

Ans

4M for correct logic & program code

```
// C program to insert an element in a linear queue using array
#include<stdio.h>
#include<conio.h>
#define n 5
void main()
{
   int queue[n],ch=1,front=0,rear=0,i,j=1,x=n;
   //clrscr();
   printf("Queue using Array");
   printf("\n1.Insertion \n2.Display \n3.Exit");
   while(ch)
   {
      printf("\nEnter the Choice:");
      scanf("%d",&ch);
      switch(ch)
      {
      case 1:
         if(rear==x)
            printf("\n Queue is Full");
         else
         {
            printf("\n Enter no %d:",j++);
            scanf("%d",&queue[rear++]);
         }
         break;
      case 2:
         printf("\n Queue Elements are:\n ");
         if(front==rear)
            printf("\n Queue is Empty");
```

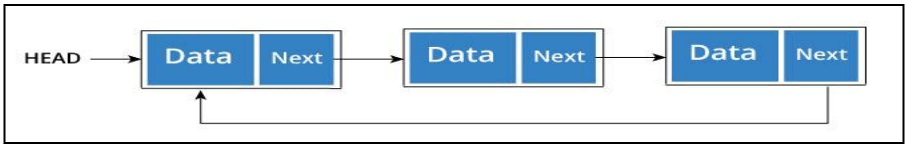
```
                else
                {
                    for(i=front; i<rear; i++)
                    {
                        printf("%d",queue[i]);
                        printf("\n");
                    }
                    break;
                case 3:
                    exit(0);
                default:
                    printf("Wrong Choice: please see the options");
                }
            }
        }
        getch();
    }
```

e **Describe circular linked list with suitable diagram. Also state advantage of circular linked list over linear linked list.** 4 M

Ans **Circular Linked List**

2M for description 1M for diagram and 1M for any one advantage

A circular linked list is a variation of linked list in which the last element is linked to the first element. This forms a circular loop.



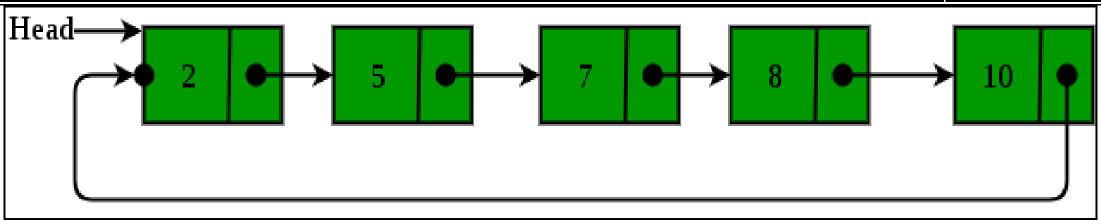
A circular linked list can be either singly linked or doubly linked.

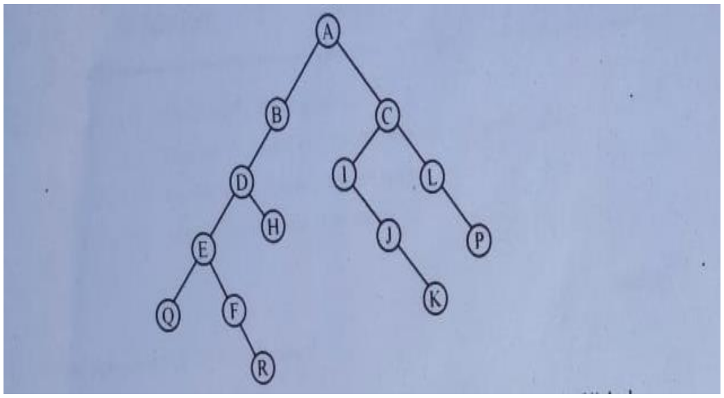
- for singly linked list, next pointer of last item points to the first item
- In doubly linked list, prev pointer of first item points to last item as well.

We declare the structure for the circular linked list in the same way as follows:

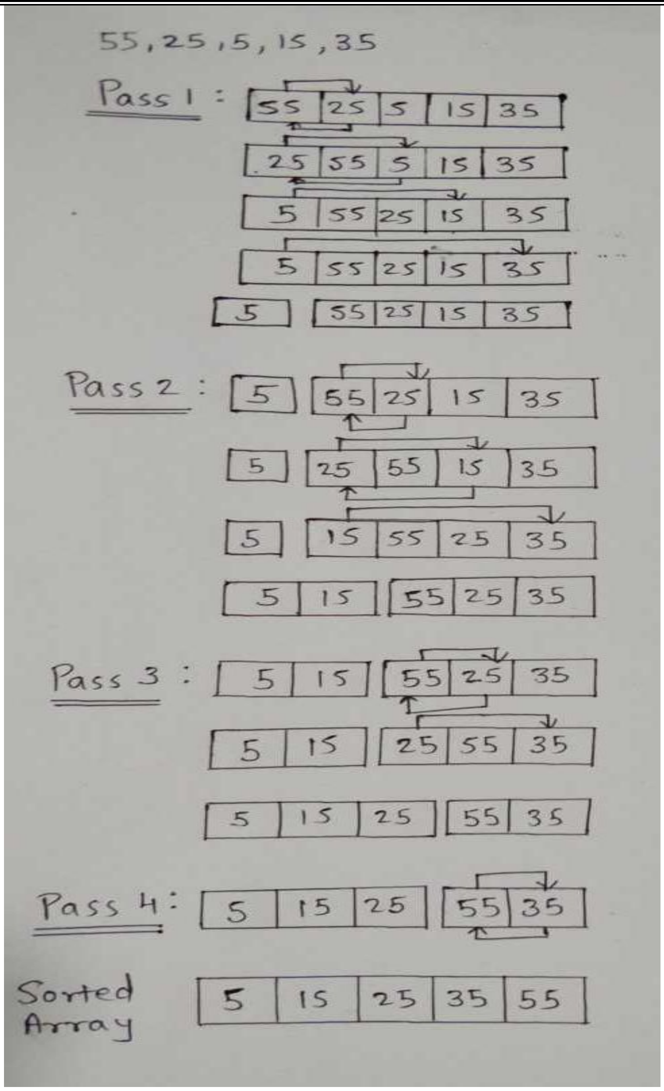
```
Struct node
{
Int data;
Struct node *next;
};
Typedef struct node *Node;
Node *start = null;
Node *last = null;
```

For example:

		 **Advantages of Circular Linked Lists:** **1)** Any node can be a starting point. We can traverse the whole list by starting from any point. We just need to stop when the first visited node is visited again. **2)** Useful for implementation of queue. Unlike this implementation, we don't need to maintain two pointers for front and rear if we use circular linked list. We can maintain a pointer to the last inserted node and front can always be obtained as next of last. **3)** Circular lists are useful in applications to repeatedly go around the list. For example, when multiple applications are running on a PC, it is common for the operating system to put the running applications on a list and then to cycle through them, giving each of them a slice of time to execute, and then making them wait while the CPU is given to another application. It is convenient for the operating system to use a circular list so that when it reaches the end of the list it can cycle around to the front of the list. **4)** Circular Doubly Linked Lists are used for implementation of advanced data structures like Fibonacci Heap.	
5		**Attempt any TWO of the following :**	**12 M**
	a	**Write algorithm for performing push and pop operations on stack.**	6 M
	Ans	**Push algorithm: -** Max is maximum size of stack. Step 1: [Check for stack full/ overflow] If stack_top is equal to max-1 then Display output as "Stack Overflow" and return to calling function Otherwise Go to step 2 Step 2: [Increment stack_top] Increment stack top pointer by one. stack_top=stack_top +1; Step 3: [Insert element] stack [stack_top] = item; Step 4: return to calling function **Pop algorithm: -** Max is maximum size of stack. Step 1: [Check for stack empty/underflow]	3marks for Push algorithm and 3marks for Pop operation

		If stack_top is equal to -1 then Display output as “Stack Underflow” and return to calling function Otherwise Go to step 2 Step 2: [delete element] stack [stack_top] = item; Step 3: [Decrement stack_top] Decrement stack top pointer by one. stack_top=stack_top -1; Step 4: return to calling function.	
	b	**For given binary tree write in-order, pre-order and post-order traversal.**  A B C D I L H E J P Q F K R	6 M
	Ans	**Inorder Traversal: Q,E,F,R,D,H,B,A,I,J,K,C,L,P** **Preorder Traversal: A,B,D,E,Q,F,R,H,C,I,J,K,L,P** **Postorder Traversal: Q,R,F,E,H,D,B,K,J,I,P,L,C,A**	2marks for each traversal
	c	**Write an algorithm to insert an element at the beginning and end of linked list.**	6 M
	Ans	**Algorithm to insert an element at the beginning of linked list:** **1.** Start **2.** Create the node pointer *temp Struct node * temp **3.** Allocate address to temp using malloc temp = malloc(sizeof(struct node)); **4.** Check whether temp is null, if null then Display “Overflow” else	3marks for each algorithm

		temp-> info=data temp-> next=start **5.** Start=temp **6.** stop **Algorithm to insert an element at the end of linked list:** **1.** Start **2.** Create two node pointers *temp, *q struct node * temp, *q; **3.** q= start **4.** Allocate address to temp using malloc temp = malloc(sizeof(struct node)); **5.** Check whether temp is null, if null then Display “Overflow” else temp-> info=data temp-> next=null **6.** While(q->next!=null) q= q-> next **7.** q->next= temp **8.** stop	
6		**Attempt any TWO of the following :**	**12 M**
	a	**Describe working of selection sort method. Also sort given input list in ascending order using selection sort input list:- 55, 25, 5, 15, 35.**	6 M
	Ans	**Working of Selection sort**: Selection Sort algorithm is used to arrange a list of elements in a particular order (Ascending or Descending). In selection sort, the first element in the list is selected and it is compared repeatedly with remaining all the elements in the list. If any element is smaller than the selected element (for ascending order), then both are swapped. Then we select the element at second position in the list and it is compared with remaining all elements in the list. If any element is smaller than the selected element, then both are swapped. This procedure is repeated till the entire list is sorted.	3marks for description, 3marks for correct solution



55, 25, 5, 15, 35

Pass 1 : | 55 | 25 | 5 | 15 | 35 |

| 25 | 55 | 5 | 15 | 35 |

| 5 | 55 | 25 | 15 | 35 |

| 5 | 55 | 25 | 15 | 35 |

| 5 | | 55 | 25 | 15 | 35 |

Pass 2 : | 5 | | 55 | 25 | 15 | 35 |

| 5 | | 25 | 55 | 15 | 35 |

| 5 | | 15 | 55 | 25 | 35 |

| 5 | 15 | | 55 | 25 | 35 |

Pass 3 : | 5 | 15 | | 55 | 25 | 35 |

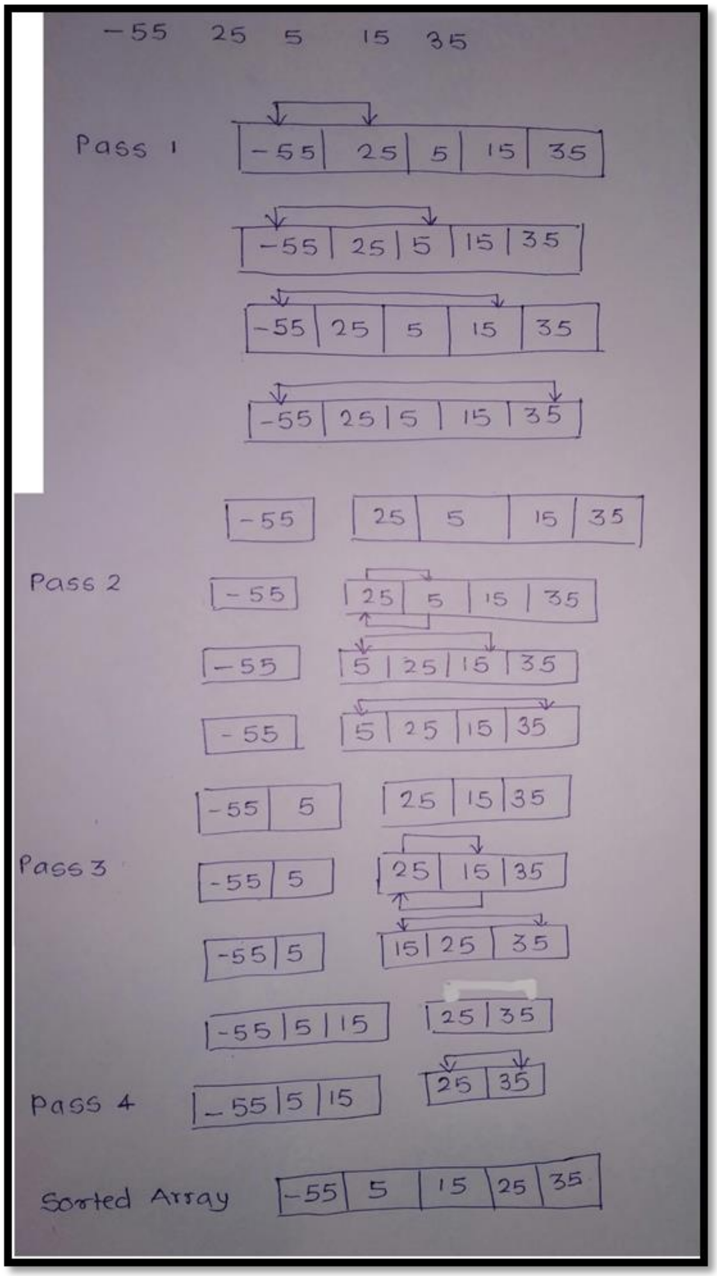
| 5 | 15 | | 25 | 55 | 35 |

| 5 | 15 | 25 | | 55 | 35 |

Pass 4 : | 5 | 15 | 25 | | 55 | 35 |

Sorted Array | 5 | 15 | 25 | 35 | 55 |

OR



-55 25 5 15 35

Pass 1

| -55 | 25 | 5 | 15 | 35 |

| -55 | 25 | 5 | 15 | 35 |

| -55 | 25 | 5 | 15 | 35 |

| -55 | 25 | 5 | 15 | 35 |

| -55 | | 25 | 5 | 15 | 35 |

Pass 2

| -55 | | 25 | 5 | 15 | 35 |

| -55 | | 5 | 25 | 15 | 35 |

| -55 | | 5 | 25 | 15 | 35 |

| -55 | 5 | | 25 | 15 | 35 |

Pass 3

| -55 | 5 | | 25 | 15 | 35 |

| -55 | 5 | | 15 | 25 | 35 |

| -55 | 5 | 15 | | 25 | 35 |

Pass 4

| -55 | 5 | 15 | | 25 | 35 |

Sorted Array

| -55 | 5 | 15 | 25 | 35 |

b **Define the term recursion. Write a program in C to display factorial of an entered number using recursion.** 6 M

Ans **Definition:** Recursion is the process of calling function by itself. A recursive function body contains function call statement that calls itself repeatedly.

2marks for definition, 4marks for correct program

Program:

```
#include<stdio.h>

#include<conio.h>

int fact(int n);

 void main()
```

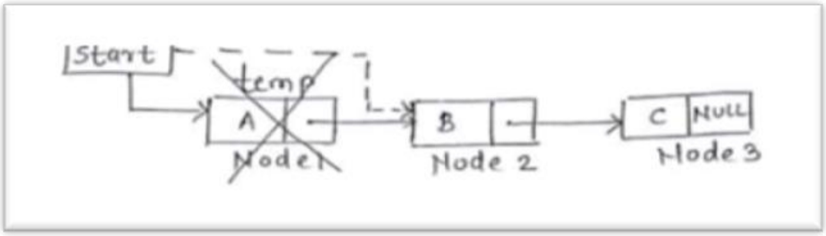
```
{
int n;
 clrscr();
 printf("\nThe factorial of % is = %d",n,fact(n));
getch();
}
 int fact(int n)
{
if(n==1)
return 1;
else
return(n*fact(n-1));
}
```

c **Describe procedure to delete an element from singly linked list using diagram.** 6 M

Ans

In a linear linked list, a node can be deleted from the beginning of list, from in between positions and from end of the list.

Delete a node from the beginning:-



Node to be deleted is node1.Create a temporary node as 'temp'. Set 'temp' node with the address of first node. Store address of node 2 in header pointer 'start' and then delete 'temp' pointer with free function. Deleting temp pointer deletes the first node from the list.

OR

Step 1: Create temporary node 'temp'.

Step 2: Assign address of first node to 'temp' pointer.

Step 3: Store address of second node (temp->next) in header pointer 'start'.

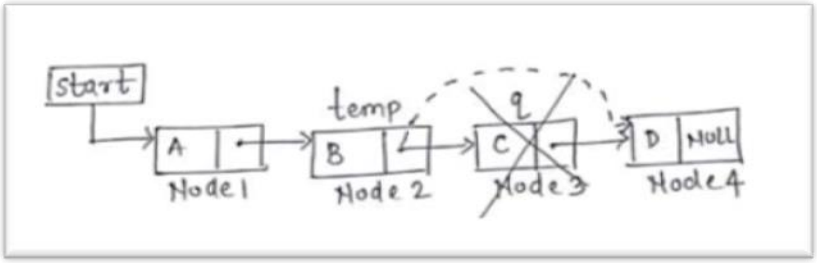
Step 4: Free temp.

Delete a node from in between position:-

**Note: Correct algorithm or program shall be considered.

Any two deletions shall be considered

3marks each



Node to be deleted is node3.Create a temporary node as ‘temp’ and ‘q’. Set ‘temp’ node with the address of first node. Traverse the list up to the previous node of node 3 and mark the next node (node3) as ‘q’. Store address from node ‘q’ into address field of ‘temp’ node. Then delete ‘q’ pointer with free function. Deleting ‘q’ pointer deletes the node 3 from the list.

OR

Step 1: Create temporary node ‘temp’, ’q’.

Step 2: Assign address of first node to ‘temp’ pointer.

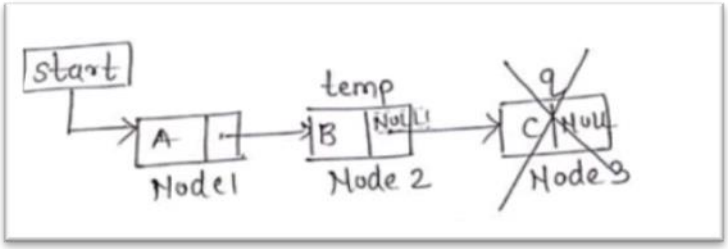
Step 3: Traverse list up to previous node of node to be deleted.

Step 4: Mark the node to be deleted ‘q’.

Step 5: Store address from node ‘q’ in address field of ‘temp’ node (temp- >next=q->next).

Step 6: Free q.

Delete a node from the end:-



Node to be deleted is node 3.Create a temporary node as ‘temp’ and ‘q’. Set ‘temp’ node with the address of first node. Traverse the list up to the second last node and mark the last node as ‘q’. Store NULL value in address field of ‘temp’ node and then delete ‘q’ pointer with free function. Deleting q pointer deletes the last node from the list.

OR

Step 1: Create temporary node ‘temp’,’q’.
Step 2: Assign address of first node to ‘temp’ pointer.
Step 3: Traverse list upto second last node.
Step 4: Mark last node’s address in node ‘q’.
Step 5: store NULL value in address field of second last node (temp->next).
Step 6: Free q