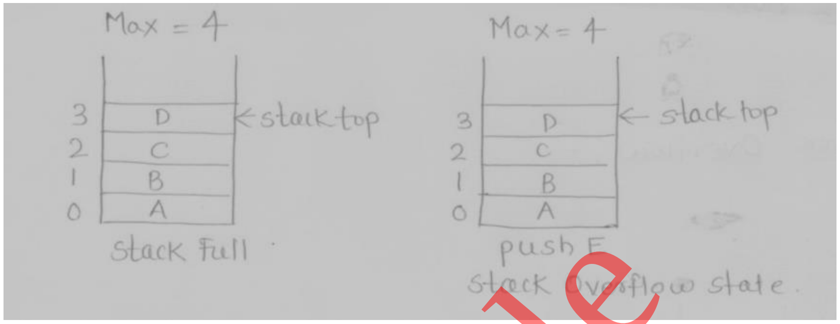
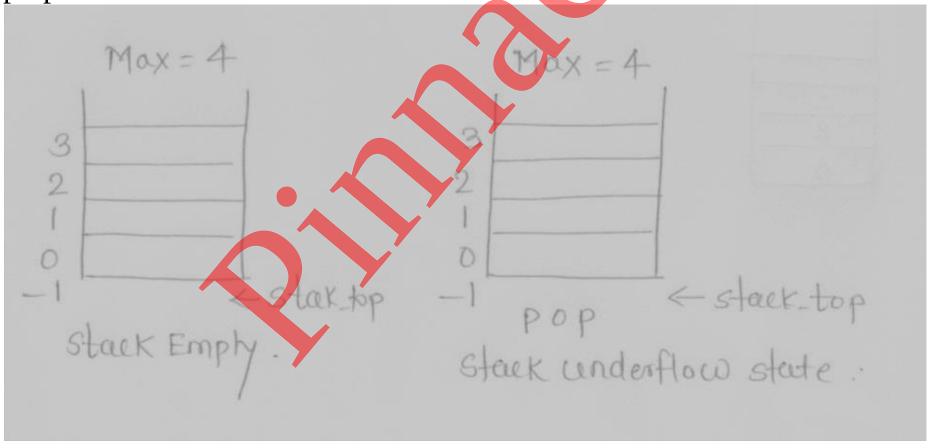
Winter – 19 EXAMINATION

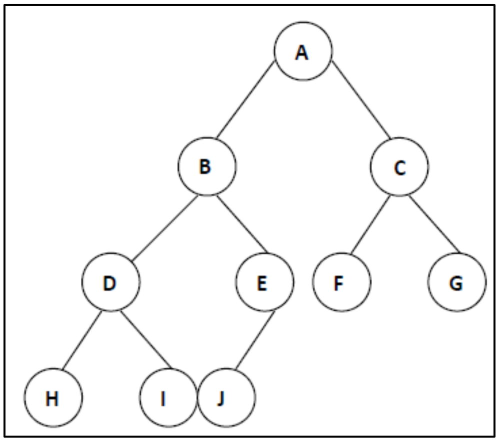
Subject Name: Data Structure Using 'C' **Model Answer** **Subject Code: 22317**

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:**	**10M**
	a	**Write any four operations that can be performed on data structure.**	**2M**
	Ans	1. **Data structure operations (Non Primitive)** 2. **Inserting:** Adding a new data in the data structure is referred as insertion. 3. **Deleting:** Removing a data from the data structure is referred as deletion. 4. **Sorting:** Arranging the data in some logical order (ascending or descending, numerically or alphabetically). 5. **Searching:** Finding the location of data within the data structure which satisfy the searching condition. 6. **Traversing:** Accessing each data exactly once in the data structure so that each data item is traversed or visited. 7. **Merging:** Combining the data of two different sorted files into a single sorted file. 8. **Copying:** Copying the contents of one data structure to another. 9. **Concatenation:** Combining the data from two or more data structure. **OR**	2 M for any 4 Operation

		Data structure operations (Primitive) 1. Creation: To create new Data Structure 2. Destroy: To delete Data Structure 3. Selection: To access (select) data from the data structure 4. Updating: To edit or change the data within the data structure.	
	b	**Define the term overflow and underflow with respect to stack.**	**2M**
	Ans	**Stack overflow**: When a stack is full and push operation is performed to insert a new element, stack is said to be in overflow state.  Max = 4 3 2 1 0 D C B A ← stack top Stack Full Max = 4 3 2 1 0 D C B A ← stack top push E Stack Overflow state. **Stack underflow**: When there is no element in a stack (stack empty) and pop operation is called then stack is said to underflow state.  Max = 4 3 2 1 0 -1 ← stack top Stack Empty. Max = 4 3 2 1 0 -1 ← stack top POP Stack underflow state.	1 M for stack overflow and 1M for stack underflow
	c	**Define the following term w.r.t. tree: (i) In-degree (ii) out-degree.**	**2M**
	Ans	**In -degree:** Number of edges coming towards node is in-degree of node. For e.g. : In degree of node B is 1 **Out -degree:** Number of edges going out from node is out -degree of node. For e.g. Out Degree of is node D is 2	1 M for each correct definition

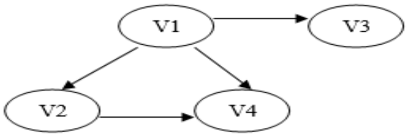
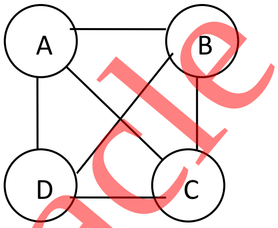
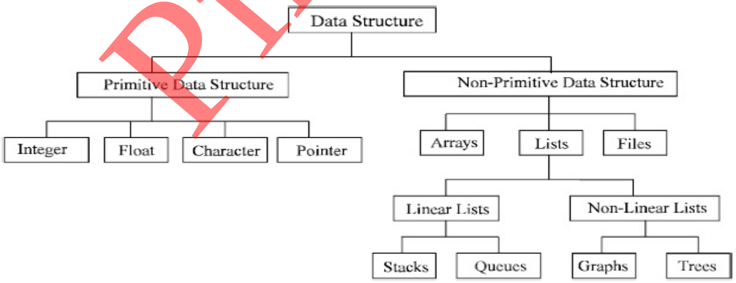


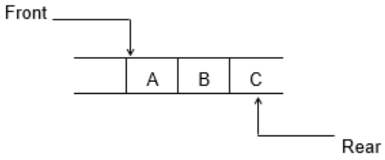
d **Evaluate the following arithmetic expression P written in postfix notation: P : 4, 2, ^, 3, *,3,-,8,4 ,/,+** **2M**

Ans

Sr. No.	Symbol Scanner	STACK
1	4	4
2	2	4, 2
3	^	16
4	3	16, 3
5	*	48
6	3	48,3
7	-	45
8	8	45,8
9	4	45,8,4
10	/	45,2
11	+	47

2 M for correct answer

	e	**Describe directed and undirected graph.**	**2M**
	Ans	**Direct Graph:** A directed graph is defined as the set of ordered pair of vertices and edges where each connected edge has **assigned a direction.**  **Undirected Graph :** An undirected graph G is a graph in which each edge e is not assigned a direction. 	1M for each definition with diagram
	f	**Give classification of data structure.**	**2M**
	Ans	 Data Structure Primitive Data Structure Non-Primitive Data Structure Integer Float Character Pointer Arrays Lists Files Linear Lists Non-Linear Lists Stacks Queues Graphs Trees	2 M for diagram
	g	**Define queue. State any two applications where queue is used.**	**2M**
	Ans	A **Queue** is an ordered collection of items. It has two ends, front and rear. Front end is used to delete element from queue. Rear end is used to insert an element in queue. Queue has two ends; the element entered first in the queue is removed first from the queue. So it is called as FIFO list.	1M for definition, 1M for applications (any two)



APPLICATIONS OF QUEUES

1. Round Robin Technique for processor scheduling is implemented using queues.

2. All types of customer service (like railway ticket reservation) center software's are designed using queues to store customer's information.

3. Printer server routines are designed using queues. A number of users share a printer using printer server (a dedicated computer to which a printer is connected), the printer server then spools all the jobs from all the users, to the server's hard disk in a queue. From here jobs are printed one-by-one according to their number in the queue.

2. **Attempt any Three of the following:** **12M**

a **Sort the given number in ascending order using Radix sort:**
348, 14, 641, 3851, 74. **4M**

Ans **Pass 1:**

4 M for correct answer

	0	1	2	3	4	5	6	7	8	9
0348									0348	
0014					0014					
0641		0641								
3851		3851								
0074					0074					

0641,3851,0014,0074,0348

Pass 2:

	0	1	2	3	4	5	6	7	8	9
0641					0641					
3851						3851				
0014		0014								
0074								0074		
0348					0348					

0014,0641,0348,3851,0074

Pass 3:

	0	1	2	3	4	5	6	7	8	9
0014	0014									
0641							0641			
0348				0348						
3851									3851	
0074	0074									

0014,0074,0348,0641,3851

Pass 4:

	0	1	2	3	4	5	6	7	8	9
0014	0014									
0074	0074									
0348	0348									
0641	0641									
3851					3851					

Sorted Elements are: 14, 74, 348, 641, 3851

b **Write an algorithm to insert a new node at the beginning and end of the singly linked list.** **4M**

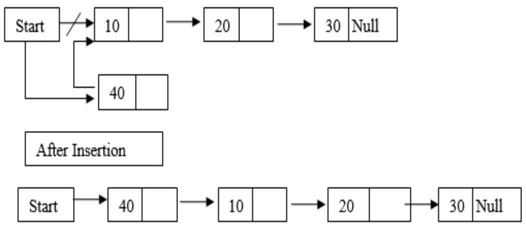
Ans

1. Algorithm for inserting a node at the beginning

```
Insert first(start, item)

1.  [check the overflow]
        if Ptr=NULL then print 'Overflow'

        exit

        else

        Ptr=(node *) malloc (size of (node))

//create new node from memory and assign its address to ptr
```

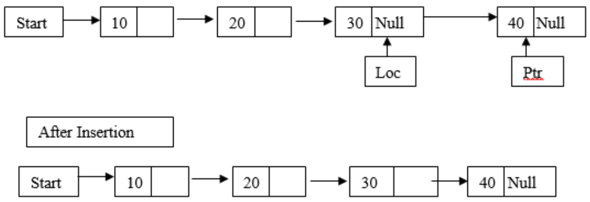
2M for Algorithm for inserting a node at the beginning
2M for Algorithm for Inserting A Node at the End

```
End if
2.  set Ptr->num = item
3.  set Ptr->next=start
4.  set start=Ptr
```



2. Algorithm for Inserting A Node at the End

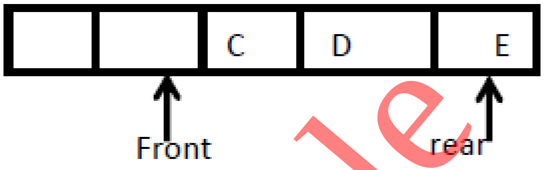
```
insert last (start, item)
1.  [check for overflow]
    If Ptr=NULL, then print 'Overflow'
    exit
    else
    Ptr=(node * ) malloc (sizeof (node));
    end if
2.  set Ptr->info=item
3.  set Ptr->next=NULL
4.  if start=NULL and if then set start=P
5.  set loc=start
6.  repeat step 7 until loc->next != NULL
7.  set loc=loc->next
8.  set loc->next=P
```



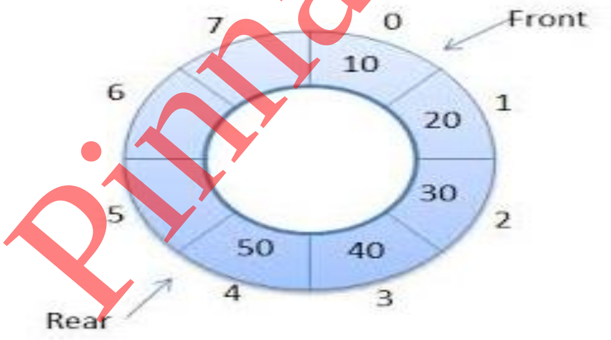
	c	**Explain the concept of circular Queue along with its need.**	4M
	Ans	• Circular queue are the queues implemented in circular form rather than in a straight line. • Circular queues overcome the problem of unutilized space in linear queue implemented as an array. • The main disadvantage of linear queue using array is that when	3 M for explanation and & 1M for need

elements are deleted from the queue, new elements cannot be added in their place in the queue, i.e. the position cannot be reused. After rear reaches the last position, i.e. MAX-1 in order to reuse the vacant positions, we can bring rear back to the 0th position, if it is empty, and continue incrementing rear in same manner as earlier.

- Thus rear will have to be incremented circularly. For deletion, front will also have to be incremented circularly.
- Rear can be incremented circularly by the following code. If ((rear == MAX-1) and (front !=0) Rear =0; Else Rear= rear +1; Example: Assuming that the queue contains three elements.



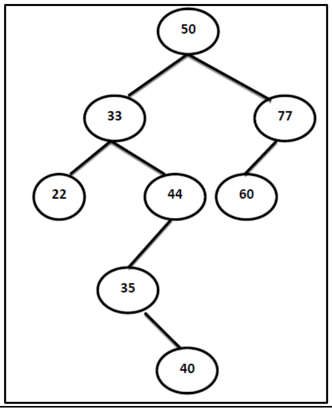
- Now we insert an element F at the beginning by bringing rear to the first position in the queue. this can be represented circularly as shown.



Need of Circular Queue:

- Circular queues overcome the problem of unutilized space in linear queue implemented as an array.
- The element can be stored efficiently in an array so as to wrap around so that the end of queue is followed by front of the queue.

	d	**Draw a binary search tree for the given number. 50, 33, 44, 22, 77, 35, 60, 40.**	**4M**
	Ans		4 M for correct answer



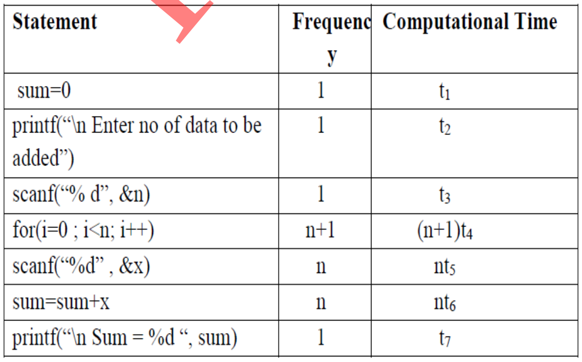
3. **Attempt any Three of the following:** **12M**

a **Explain time and space complexity with an example.** **4M**

Ans **Time Complexity:** Time complexity of program or algorithm is amount of computer time that it needs to run to completion. To measure time complexity of an algorithm we concentrate on developing only frequency count for key statements.

2M for Time Complexity and
2M for space complexity

```
Example:
#include<stdio.h>
void main ()
{
int i, n, sum, x;
sum=0;
printf(“\n Enter no of data to be added”);
scanf(“% d”, &n);
for(i=0 ; i<n; i++)
```



Statement	Frequency	Computational Time
sum=0	1	t_1
printf(“\n Enter no of data to be added”)	1	t_2
scanf(“% d”, &n)	1	t_3
for(i=0 ; i<n; i++)	n+1	$(n+1)t_4$
scanf(“%d” , &x)	n	nt_5
sum=sum+x	n	nt_6
printf(“\n Sum = %d “, sum)	1	t_7

Total computational time= t1+t2+t3+(n+1)t4 +nt6+nt5+t7
T= n(t4+t5+t6)+ (t1+t2+t3+t4+t7)
For large n , T can be approximated to
T= n(t4+t5+t6)= kn where k= t4+t5+t6
Thus T = kn or

Space Complexity: Total amount of computer memory required by an algorithm to complete its execution is called as space complexity of that algorithm. When a program is under execution it uses the computer memory for THREE reasons. They are as follows...

- Instruction Space: It is the amount of memory used to store compiled version of instructions.
- Environmental Stack: It is the amount of memory used to store information of partially executed functions at the time of function call.
- Data Space: It is the amount of memory used to store all the variables and constants.

If the amount of space required by an algorithm is increased with the increase of input value, then that space complexity is said to be Linear Space Complexity.

Example:

```
int sum(int A[ ], int n)
{
  int sum = 0, i;
  for(i = 0; i < n; i++)
    sum = sum + A[i];
  return sum;}
```

In the above piece of code it requires

'n*2' bytes of memory to store array variable 'a[ ]'
2 bytes of memory for integer parameter 'n'
4 bytes of memory for local integer variables 'sum' and 'i' (2 bytes each)
2 bytes of memory for return value.

That means, totally it requires '2n+8' bytes of memory to complete its execution. Here, the total amount of memory required depends on the value of 'n'. As 'n' value increases the space required also increases proportionately. This type of space complexity is said to be **Linear Space Complexity.**

OR

Time complexity:- Time complexity of a program/algorithm is the amount of computer time that it needs to run to completion. While calculating time complexity, we develop frequency count for all key statements which are important and basic instructions of an algorithm.

Example: Consider three algorithms given below:-

		Algorithm A: - a=a+1 Algorithm B: - for x = 1 to n step 1 a=a+1 Loop Algorithm C:- for x=1 to n step 1 for y=1 to n step 1 a=a+1 Loop Frequency count for algorithm A is 1 as a=a+1 statement will execute only once. Frequency count for algorithm B is n as a=a+1 is key statement executes n time as the loop runs n times. Frequency count for algorithm C is n as a=a+1 is key statement executes n2 time as the inner loop runs n times, each time the outer loop runs and the outer loop also runs for n times. **Space complexity**:- Space complexity of a program/algorithm is the amount of memory that it needs to run to completion. The space needed by the program is the sum of the following components:- **Fixed space requirements**: - It includes space for instructions, for simple variables, fixed size structured variables and constants. **Variable time requirements**: - It consists of space needed by structured variables whose size depends on particular instance of variables. Example: - additional space required when function uses recursion.	
	b	**Convert the following infix expression to postfix expression using stack and show the details of stack in each step.((A+B)*D)^(E-F)**	**4M**
	Ans	**infix expression:** **(((A+B)*D)^(E-F))**	Correct answer-4M

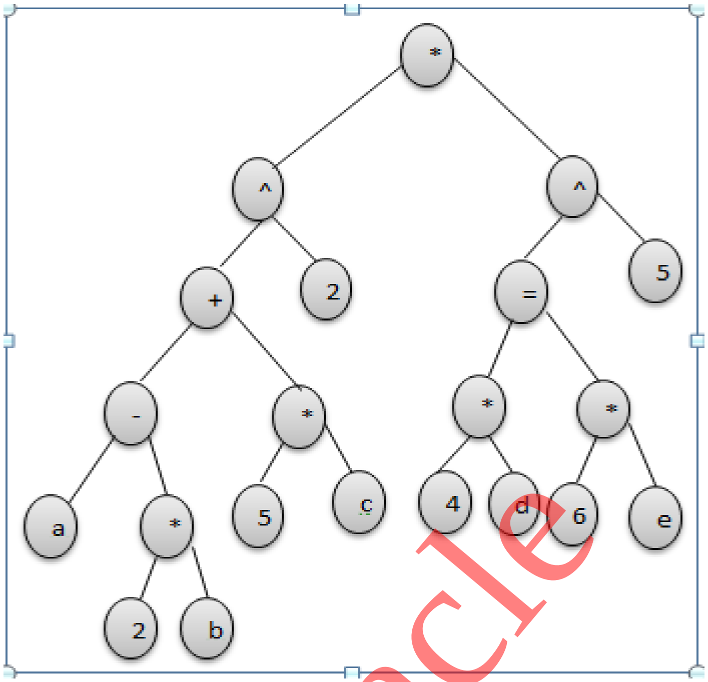
Current Symbol	Operator Stack	Postfix array
(	(	Empty
(	((	Empty
(	(((	Empty
A	(((	A
+	(((+	A
B	(((+	AB
)	((	AB+
*	((*	AB+
D	((*	AB+D
)	(	AB+D*
^	(^	AB+D*
(	(^(	AB+D*
E	(^(	AB+D*E
-	(^(-	AB+D*E
F	(^(-	AB+D*EF
)	(^	AB+D*EF-
)	EMPTY STACK	AB+D*EF-^

Postfix expression: AB+D*EF-^

	c	**Implement a 'C' program to search a particular data from the given array using Linear Search.**	**4M**
	Ans	**Program:-**	

		# include<stdio.h>
		#include <conio.h>
		void main ()
		{
		int a[10], n, key,i,c=0;
		clrscr( );
		printf (“Enter number of array elements\n”);
		scanf (“%d”, &n);
		printf (“Enter array elements\n”);
		for (i=0; i< n; i++)
		scanf (“%d”, &a[i]);
		prinntf (“Enter key value\n”);
		scanf (“%d”, &key);
		
		for(i=0;i<n-1;i++)
		{
		
		if (key == a[i])
		{
		c=1;
		printf (“%d is found at location %d\n”, key, i+1);
		break;
		}
		
		}
		if (c==0)
		printf (“%d not present in the list\n”,key);
		getch();
		}
			2M for logic And 2 M for syntax

	d	**Draw an expression tree for the following expression:** **$(a-2b+5e)^{2} * (4d=6e)^{5}$.**	**4M**
	Ans		Correct Expression tree-4M



4. **Attempt any Three of the following:** **12M**

a **Find the position of element 21 using binary search method in array 'A' given below: A=(11,5,21,3,29,17,2,45}** **4M**

Ans **Given Array**

Each correct step -2M each

11	**5**	**21**	**3**	**29**	**17**	**2**	**45**

Sorted Array for input:

2	**3**	**5**	**11**	**17**	**21**	**29**	**45**

Key element to be searched=21

Step1

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
2	**3**	**5**	**11**	**17**	**21**	**29**	**45**

l=0 and u=n-1 =7

mid=(l+u)/2 = 7/2 = 3

a[mid]=11 not equal to 21
and

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

21 > 11 l=mid+1 = 4 and u = 7

Step 2:

<table>
<tr><td><b>4</b></td><td><b>5</b></td><td><b>6</b></td><td><b>7</b></td></tr>
<tr><td><b>17</b></td><td><b>21</b></td><td><b>29</b></td><td><b>45</b></td></tr>
</table>

l=4 and u =7

mid= 11/2 = 5

a[mid]=21 equal to key element 21

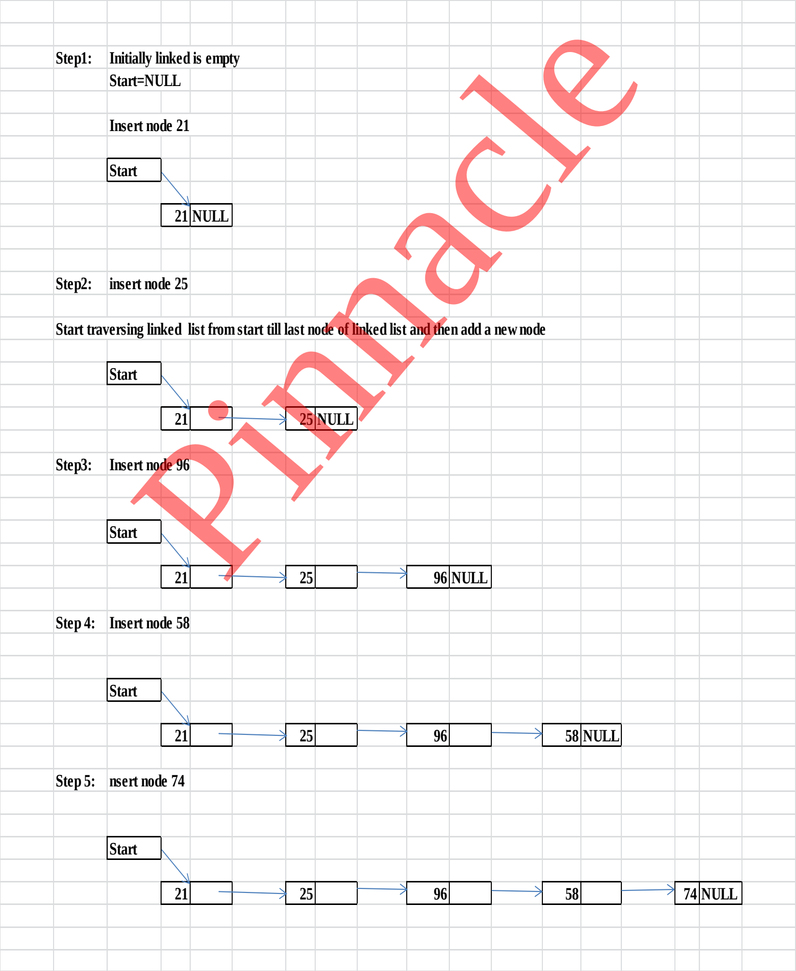
therefore key element 21 is fount un array at position 6

</td><td></td></tr>
<tr><td></td><td><b>b</b></td><td><b>Difference between tree and graph(Any 4 points)</b></td><td><b>4M</b></td></tr>
<tr><td></td><td><b>Ans</b></td><td>

<table>
<tr><th>Tree</th><th>Graph</th></tr>
<tr><td>Tree is special form of graph i.e. minimally connected graph and having only one path between any two vertices.</td><td>In graph there can be more than one path i.e. graph can have uni-directional or bi-directional paths (edges) between nodes</td></tr>
<tr><td>Tree is a special case of graph having no loops, no circuits and no self-loops.</td><td>Graph can have loops, circuits as well as can have self-loops.</td></tr>
<tr><td>Tree traversal is a kind of special case of traversal of graph. Tree is traversed in Pre-Order, In-Order and Post-Order</td><td>Graph is traversed by DFS: Depth First Search and in BFS : Breadth First Search algorithm</td></tr>
<tr><td>Different types of trees are: Binary Tree, Binary Search Tree, AVL tree, Heaps.</td><td>There are mainly two types of Graphs: Directed and Undirected graphs.</td></tr>
</table>

</td><td>Any correct 4 points- 4M</td></tr>
</table>

Tree applications: sorting and searching like Tree Traversal & Binary Search.	Graph applications : Coloring of maps, in OR (PERT & CPM), algorithms, Graph coloring, job scheduling, etc.
Tree always has n-1 edges.	In Graph, no. of edges depends on the graph.
Tree is a hierarchical model.	Graph is a network model.

	C	**Construct a singly linked list using data fields 21 25 96 58 74 and show procedure step-by-step with the help of diagram start to end.**	**4M**
	Ans	 Step1: Initially linked is empty Start=NULL Insert node 21 Step2: insert node 25 Start traversing linked list from start till last node of linked list and then add a new node Step3: Insert node 96 Step 4: Insert node 58 Step 5: nsert node 74 Start 21 NULL Start 21 25 NULL Start 21 25 96 NULL Start 21 25 96 58 NULL Start 21 25 96 58 74 NULL	correct construction - 3M and explaination- 1M

	d	**Show the effect of PUSH and POP operation on the stack of size 10.**
		PUSH(10)
		PUSH(20)
			4M

<table>
<tr><td></td><td></td><td><b>POP</b>
<b>PUSH(30)</b></td><td></td></tr>
<tr><td></td><td><b>Ans</b></td><td>
<b>Initial Stack empty</b>
<table>
<tr><td></td><td>stack[9]</td><td></td></tr>
<tr><td></td><td>stack[8]</td><td></td></tr>
<tr><td></td><td>stack[7]</td><td></td></tr>
<tr><td></td><td>stack[6]</td><td></td></tr>
<tr><td></td><td>stack[5]</td><td></td></tr>
<tr><td></td><td>stack[4]</td><td></td></tr>
<tr><td></td><td>stack[3]</td><td></td></tr>
<tr><td></td><td>stack[2]</td><td></td></tr>
<tr><td></td><td>stack[1]</td><td></td></tr>
<tr><td></td><td>stack[0]</td><td>top= -1</td></tr>
</table>
<b>Step 1:</b>
PUSH(0)
top=top+1 stack[0]=10
<table>
<tr><td></td><td>stack[9]</td><td></td></tr>
<tr><td></td><td>stack[8]</td><td></td></tr>
<tr><td></td><td>stack[7]</td><td></td></tr>
<tr><td></td><td>stack[6]</td><td></td></tr>
<tr><td></td><td>stack[5]</td><td></td></tr>
<tr><td></td><td>stack[4]</td><td></td></tr>
<tr><td></td><td>stack[3]</td><td></td></tr>
<tr><td></td><td>stack[2]</td><td></td></tr>
<tr><td></td><td>stack[1]</td><td></td></tr>
<tr><td>10</td><td>stack[0]</td><td>top=0</td></tr>
</table>
<b>Step 2:</b>
PUSH(0)
top=top+1 stack[1]=20
<table>
<tr><td></td><td>stack[9]</td><td></td></tr>
<tr><td></td><td>stack[8]</td><td></td></tr>
<tr><td></td><td>stack[7]</td><td></td></tr>
<tr><td></td><td>stack[6]</td><td></td></tr>
<tr><td></td><td>stack[5]</td><td></td></tr>
<tr><td></td><td>stack[4]</td><td></td></tr>
<tr><td></td><td>stack[3]</td><td></td></tr>
<tr><td></td><td>stack[2]</td><td></td></tr>
<tr><td>20</td><td>stack[1]</td><td>top=1</td></tr>
<tr><td>10</td><td>stack[0]</td><td></td></tr>
</table>
<b>Step 3:</b>
POP
</td><td><b>Each correct step-1M</b></td></tr>
</table>

top=top-1 20 is deleted

	stack[9]	
	stack[8]	
	stack[7]	
	stack[6]	
	stack[5]	
	stack[4]	
	stack[3]	
	stack[2]	
	stack[1]	
10	stack[0]	top=0

Step 4:

PUSH(0)

top=top+1 stack[1]=30

	stack[9]	
	stack[8]	
	stack[7]	
	stack[6]	
	stack[5]	
	stack[4]	
	stack[3]	
	stack[2]	
30	stack[1]	top=1
10	stack[0]	

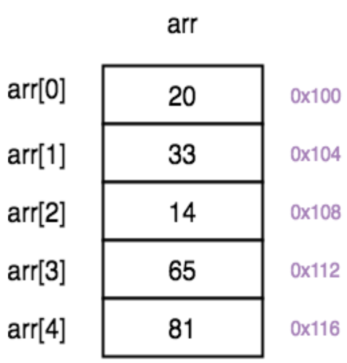
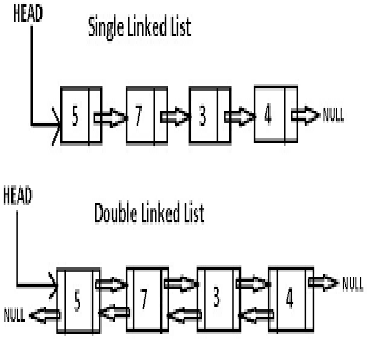
e **Compare Linked List and Array (any 4 points).** **4M**

Ans

Linked List	Array
Array is a collection of elements of similar data type.	Linked List is an ordered collection of elements of same type, which are connected to each other using pointers.
Array supports Random Access, which means elements can be accessed directly using their index, like arr[0] for 1st element, arr[6] for 7th element etc.	Linked List supports Sequential Access, which means to access any element/node in a linked list; we have to sequentially traverse the complete linked list, up to that element.

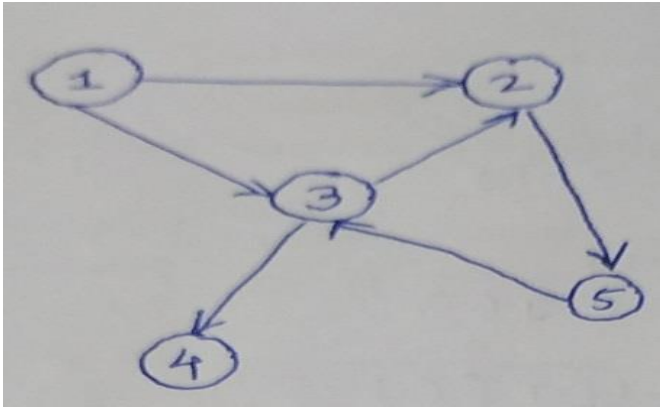
1M for each valid difference

			Hence, accessing elements in an array is fast with a constant time complexity of O (1).	To access nth element of a linked list, time complexity is O (n).		
			In array, Insertion and Deletion operation takes more time, as the memory locations are consecutive and fixed.	In case of linked list, a new element is stored at the first free and available memory location, with only a single overhead step of storing the address of memory location in the previous node of linked list. Insertion and Deletion operations are fast in linked list.		
			Memory is allocated as soon as the array is declared, at compile time. It's also known as Static Memory Allocation.	Memory is allocated at runtime, as and when a new node is added. It's also known as Dynamic Memory Allocation.		
			In array, each element is independent and can be accessed using it's index value	In case of a linked list, each node/element points to the next, previous, or maybe both nodes.		
			Array can single dimensional, two dimensional or multidime nsional	Linked list can be Linear (Singly), Doubly or Circular li nked list.		
			Size of the array must be specified at time of array declaration.	Size of a Linked list is variable. It grows at runtime, as more nodes are added to it.		
			Array gets memory allocated in the Stack section	Whereas, linked list gets memory allocated in Heap section.		

		 arr arr[0] 20 0x100 arr[1] 33 0x104 arr[2] 14 0x108 arr[3] 65 0x112 arr[4] 81 0x116 Array representation  HEAD Single Linked List 5 7 3 4 NULL HEAD Double Linked List NULL 5 7 3 4 NULL **Linked list presentation**	
5.		**Attempt any Three of the following:**	**12- M**
	a	**Implement a ‘C’ program to insert element into the queue and delete the element from the queue.**	**6M**

	Ans	#include<stdio.h>
		#include<conio.h>
		#define max 5
		void main()
		{
		int a[max],front,rear,no,ch,i;
		clrscr();
		front=rear=-1;
		do
		{
		printf("\n 1.INSERT");
		printf("\t 2.DELETE");
		printf("\t 3.EXIT");
		printf("\n\n ENTER YOUR CHOICE:- ");
		scanf("%d",&ch);
		switch(ch)
		{
		case 1:
		printf("\n ENTER ITEM TO BE INSERTED :- ");
		scanf("%d",&no);
		if(rear==max-1)
		{
		printf ("\n QUEUE IS FULL.");
			Insert logic-3M, delete logic-3M

		break;
		}
		rear=rear+1;
		a[rear]=no;
		if(front==-1)
		front=0;
		break;
		case 2:
		if(front==-1)
		{
		printf ("\n QUEUE IS EMPTY.");
		break;
		}
		no=a[front];
		printf("\n DELETED ELEMENT IS:- %d",no);
		if(front==rear)
		front=rear=-1;
		else
		front=front+1;
		break;
		case 3:
		exit(0);
		}
		printf("\n\n DO YOU WANT TO CONTINUE:(1 FOR YES/2 FOR NO):-");
		scanf("%d",&ch);
		}while(ch==1);
		getch();
		}	

	b	**Consider the graph given in following figure and answer given questions.**  **1)All simple path from 1 to 5** **2)In-degree of and out-degree of 4** **3) Give Adjacency matrix for the given graph.** **4) Give Adjacency list representation of the given graph.**	**6M**

Ans

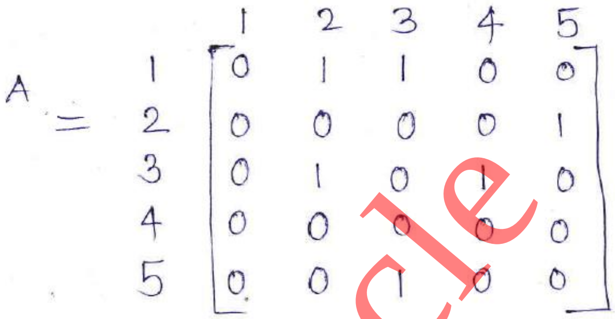
i) Nodes: **1-2-5**

ii) Nodes: **1-3-2-5**

2)

In degree of node 4- **1**, **Out degree** of node 4 - **0**

3)**Correct adjacency matrix:**



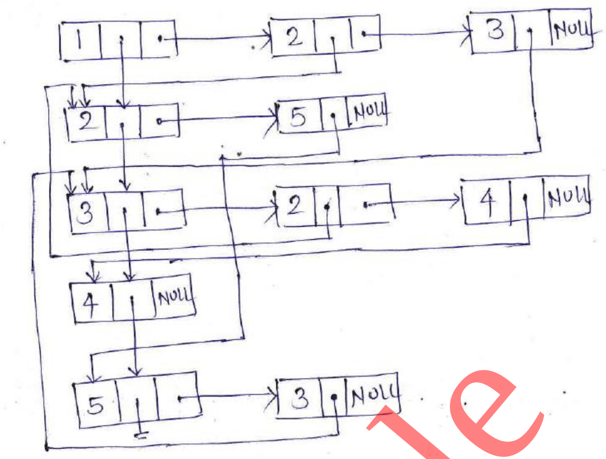
$$A = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 1 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 1 \\ 3 & 0 & 1 & 0 & 1 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 1 & 0 & 0 \end{array}$$

4) Adjacency list representation

Node	Adjacent nodes
1	2,3
2	5
3	2,4
4	NIL
5	3

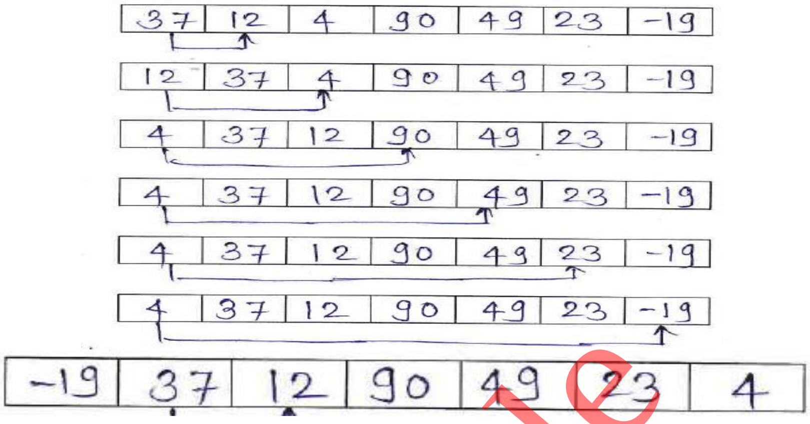
Simple path: - Each path ½ M
Each degree ½ M

Correct adjacency matrix: 2M
Adjacency list representation -2M

		Representation:  1 2 3 NULL 2 5 NULL 3 2 4 NULL 4 NULL 5 3 NULL	
	c	**Write an algorithm to search a particular node in the give linked list.**	**6M**
	Ans	**Assumption:** Node contains two fields: info and next pointer start pointer : Header node that stores address of first node step 1: start step 2: Declare variable no, flag and pointer temp step 3: Input search element step 4: Initialize pointer temp with the address from start pointer.(temp=start), flag with 0 step 5: Repeat step 6 till temp != NULL step 6: compare: temp->info = no then set flag=1 and go to step 7 otherwise increment pointer temp and go to step5 step 7: compare: flag=1 then display "Node found" otherwise display "node not found" step 8: stop	Correct steps of algorithm-6M
6.		**Attempt any Three of the following:**	**12M**
	a	**Elaborate the steps for performing selection sort for given elements of array. A={37,12,4,90,49,23,-19}**	**6M**

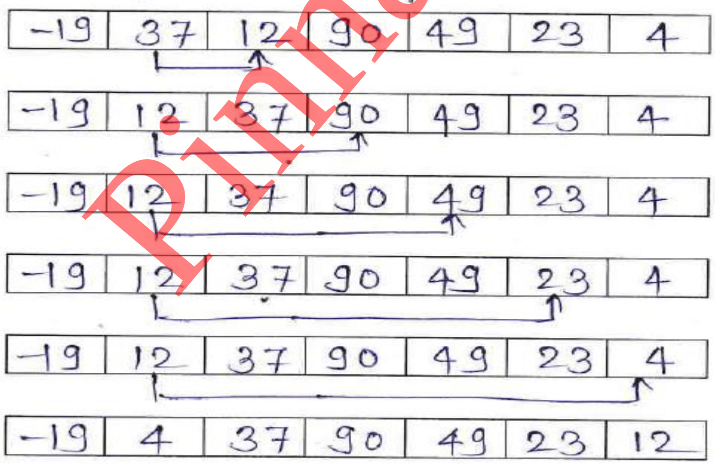
<table>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>Ans</td><td>

Pass 1



37 12 4 90 49 23 -19
12 37 4 90 49 23 -19
4 37 12 90 49 23 -19
4 37 12 90 49 23 -19
4 37 12 90 49 23 -19
4 37 12 90 49 23 -19
-19 37 12 90 49 23 4

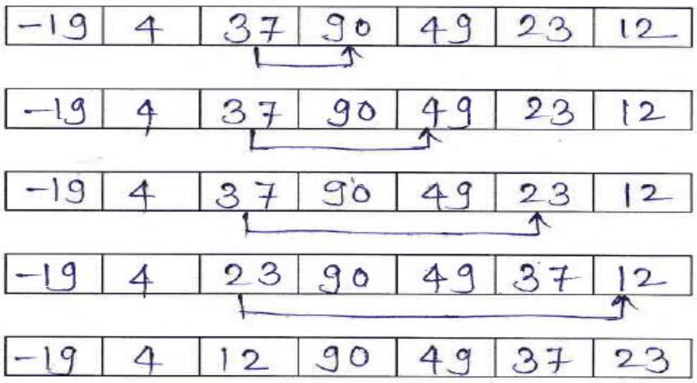
Pass 2



-19 37 12 90 49 23 4
-19 12 37 90 49 23 4
-19 12 37 90 49 23 4
-19 12 37 90 49 23 4
-19 12 37 90 49 23 4
-19 4 37 90 49 23 12

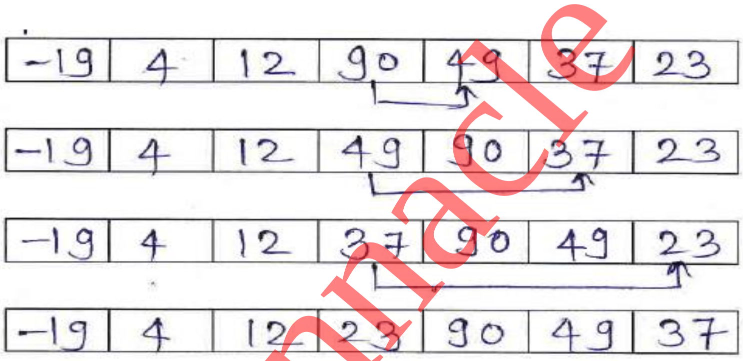
</td><td>Correct steps: each pass-1M</td></tr>
</table>

Pass 3



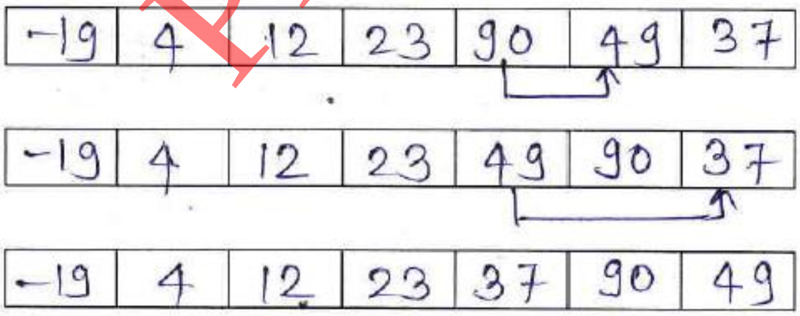
-19	4	37	90	49	23	12
-19	4	37	90	49	23	12
-19	4	37	90	49	23	12
-19	4	23	90	49	37	12
-19	4	12	90	49	37	23

Pass 4



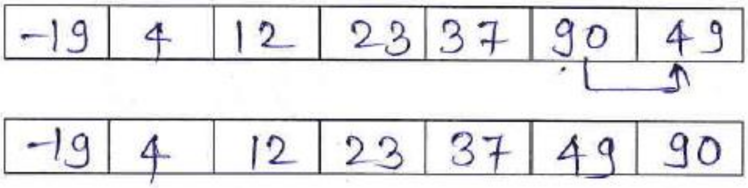
-19	4	12	90	49	37	23
-19	4	12	49	90	37	23
-19	4	12	37	90	49	23
-19	4	12	23	90	49	37

Pass 5



-19	4	12	23	90	49	37
-19	4	12	23	49	90	37
-19	4	12	23	37	90	49

Pass 6



-19	4	12	23	37	90	49
-19	4	12	23	37	49	90

	b	**Explain the concept of recursion using stack.**	**6M**
	Ans	Recursion is a process of calling a function by itself. a recursive function body contains a function call statement that calls itself repetitively. Recursion is an application of stack. When a recursive function calls itself from body, stack is used to store temporary data handled by the function in every iteration.	Explanation- 4M & 2M for Example

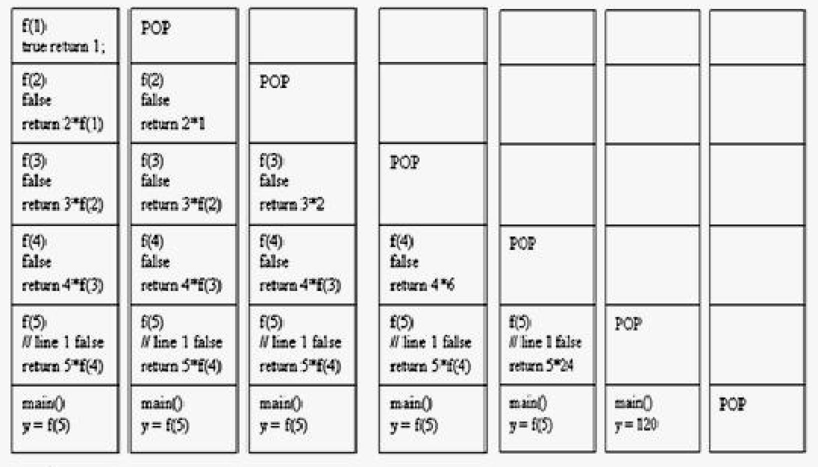
Example:

function call from main() : fact(n); // consider n=5

Function definition:

```
int fact(int n)
{
if(n==1)
return 1;
else
return(n*fact(n-1));
}
```

In the above recursive function a function call fact (n-1) makes a recursive call to fact function. Each time when a function makes a call to itself, it save its current status in stack and then executes next function call. When fact ( ) function is called from main function, it initializes n with 5. Return statement inside function body executes a recursive function call. In this call, first value of n is stored using push ( ) operation in stack (n=5) and a function is called again with value 4(n-1). In each call, value of n is push into the stack and then it is reduce by 1 to send it as argument to recursive call. When a function is called with n=1, recursive process stops. At the end all values from stack are retrieved one by one using pop ( ) operation to perform multiplication to calculate factorial of number.



f(1)
true return 1;
	POP					

f(2)
false
return 2*f(1)
	f(2)
	false
	return 2*1
		POP				

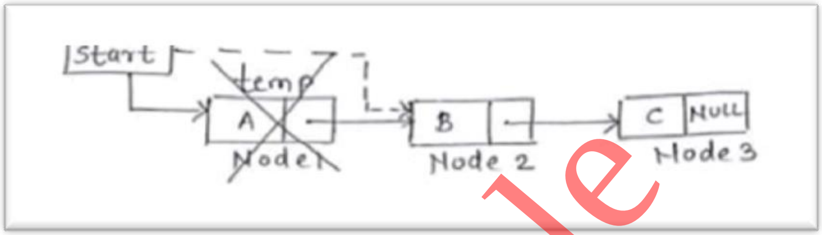
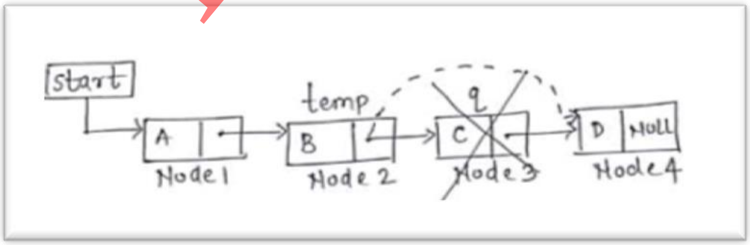
f(3)
false
return 3*f(2)
	f(3)
	false
	return 3*f(2)
		f(3)
		false
		return 3*2
			POP			

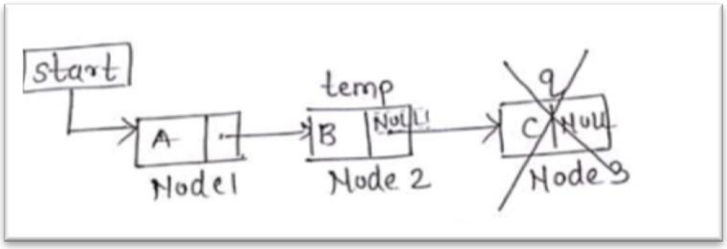
f(4)
false
return 4*f(3)
	f(4)
	false
	return 4*f(3)
		f(4)
		false
		return 4*f(3)
			f(4)
			false
			return 4*6
				POP		

f(5)
// line 1 false
return 5*f(4)
	f(5)
	// line 1 false
	return 5*f(4)
		f(5)
		// line 1 false
		return 5*f(4)
			f(5)
			// line 1 false
			return 5*f(4)
				f(5)
				// line 1 false
				return 5*24
					POP	

main() y = f(5)	main() y = f(5)	main() y = f(5)	main() y = f(5)	main() y = f(5)	main() y = 120	POP

In the above diagram, first column shows result of push operation after each

		recursive call execution. Next columns shows result of pop operation for calculating factorial.	
	c	**Show with suitable diagrams how to delete a node from singly linked list at the beginning, in between and at the end of the list.**	**6M**
	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:-**  Start temp A B C NULL Node 1 Node 2 Node 3 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. **Delete a node from in between position:-**  Start temp q A B C D NULL Node 1 Node 2 Node 3 Node 4 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.	Diagram for beginning-2M, end-2M, inbetween-2M

		Delete a node from the end:-  start temp q A B NULL C NULL Node1 Node 2 Node 3 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.	