

Note : Attempt any **FIVE** questions. All questions carry equal marks.

Q. 1. (a) What is structural programming? Discuss the advantages of structural programming paradigm.

Ans. Data may be organised in many different ways; the logical or mathematical model of a particular organisation of data is called structural programming. The choice of a particular data model depends on two considerations.

First, it must be rich enough in structure to mirror the actual relationships of the data in the real world. On the other hand, the structure should be simple enough than one can effectively process the data when necessary.

Arrays : The simplest type of data structure is a linear (one dimensional) array. By a linear array, we mean a list of a finite number n of similar data elements referenced respectively by a set of n consecutive numbers, usually 1, 2, 3,....., n . If we choose the name A for the array, then the elements of A are denoted by subscript notation.

$a_1, a_2, a_3, \dots, a_n$.

Linked Lists : It will be introduced by means of an example. Suppose a brokerage firm maintains a file where each record contains a customer's name & his or her salesperson & suppose the file contains the data rep. in table.

1. Adams Smith
2. Brown Ray
3. Clark Jones

Although the terms 'pointer' and 'link' are usually used synonymously, we use 'pointer' more often.

Tree : Data frequently contain a hierarchical relationship between various elements. The data structure which reflects this relationship is called a rooted tree graph or simply a tree. Trees will be defined basic form.

Stacks : Also called a last-in-first out (LIFO) system, is a linear list in which insertions and deletions can take place only at one end, called the top. This structure is similar in its operation to stack of cities in a spring system.

Queue : Also called first in first out system (FIFO), is a linear list in which deletions can take place only at one end of the list, the 'front' and insertions can take place only at 'rear'. This structure operates in much the same way as a line of people waiting a bus stop.

Graph : Data sometimes contain a relationship between pairs of elements which is not necessarily hierarchical in nature. For example, suppose an airline files only between the cities connected by lines. The data structure which reflects this type of relationship is called a graph.

Q. 1. (b) Discuss the properties of Array data types. Is array a structured data type? Write some important features of a C array.

Ans. Linear Arrays : A linear array is a list of a finite number n of homogeneous data elements, such that :

- (a) The elements of the array are referenced respectively by an index set consisting of n consecutive

numbers.

(b) The elements of the array are stored respectively in successive memory locations.

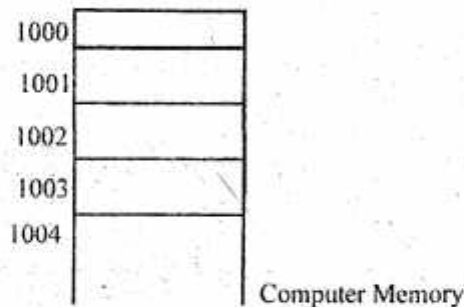
The number n of elements is called the length or size of the array. If not explicitly stated, is assumed that index set consists of the integers $1, 2, \dots, n$. In general, the length or the number of data elements of an array can be obtained from the index set by the p must.

$$\text{Length} = \text{VB} - \text{LB} + 1$$

Where VB is the largest index, called the upper bound and LB is the smaller index, called lower bound.

$$\text{Length} = \text{VB}, \text{ when } \text{LB} = 1.$$

Let LA be a linear array in the memory of the computer. The memory of the computer is simply a sequence of addressed locations.



$$\text{LOC}(\text{LA}(K)) = \text{Address of the member element } \text{LA}[K] \text{ of the array } \text{LA}$$

Accordingly, the computer does not need to keep track of the address of every element of LA, but needs to keep track only of the address of the first element of LA, denoted by :

Base(LA)

and called the base address of LA.

Array is a structured data type.

Furthermore, gives subscript K , one can locate the access the content of $\text{LA}[K]$ without scanning any other element of LA.

Q. 2. (a) Explain the pointer data type in C. Discuss the importance of pointers in C programming, giving suitable examples.

Ans. Let DATA be any array. A variable P is called a pointer if P 'points' to an element in DATA. i.e., if P contains the address of an element in DATA. Analogously, an array PTR is called a pointer array if each elements of PTR is a pointer.

Pointers and pointer arrays are used to facilitate the processing of the information in DATA.

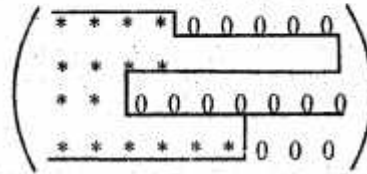
Example : Consider an organisation which divides its membership list into 4 groups, where each group contains an alphabetizes list of those members living in a certain area as :

Group 1	Group 2	Group 3	Group 4
Evans	wnraw	Davis	Baker
Harris	felt	Segal	Copper
Lewis	glass		Ford
Shaw	Hill		Gray
	King		Jones

Peny
Silver
Troy
Wagner
Red

Suppose the membership is to be stored in memory keeping track of the different groups.

One way to do this is to use a 2 dimensional $4 \times n$ array where each row contains a group, or to use a two dimensional $n \times 4$ array where each columns contains a group.



Jagged Array

Pointer Arrays : The 2 space-efficient data structures can be easily modified so that the individual groups can be indexed. This is accomplished by using a pointer array which contains the locations of the different groups or more specifically, the locations of the first elements in the different groups.

Q. 2. (b) Explain the necessity of dynamic memory allocations. Discuss any one method used in dynamic memory allocation management.

Ans. Dynamic Memory Allocation : When we declare an array in a program, the compiler allocates memory to hold the array prior to providing it. Sometimes it is required to change the size of the array and in than case, it is required to edit and compile the program.

To reduce the number of changes, it is required to allocate required memory during the run time. For example, suppose we want to read a file into the memory from the disk, which contains information about students of a university.

If we have prior information about the number of students and information associated with each student, then we can simply allocate required memory to an array, and size of array is equal to the size of file. If the size is not known and we have to load it into memory and defining very large array, so that file can fit into the array.

If the size of the array is larger than that of file, then the space wasted.

If the size of file is larger than it is not possible to load the complete file into the array, leading to a need for dynamic memory allocation.

In the dynamic memory allocation, we use a pointer variable and its size is allocated during the execution of the program.

There are 2 principal functions in C++ which are used to allocate required memory to pointer variable.

(a) malloc ()

(b) calloc ()

Both malloc () and calloc () functions perform the operations & checks memory availability based on memory block size requested.

Example :

```
// calloc function
// calloc.cpp.
#include <iostream.h>
#include <malloc.h>
```

```
# include <stdlib.h>
# include <stdio.h>
void main()
{
    int * p;
    if (p = (int *) calloc (800, 2)) == NULL)
    {
        cout << "\n out of memory \n";
        exit(0);
    }
    for (int i = 0; i < 800; i++)
    {
        cout << " " << (p+i);
        cout << " " << * (p + i);
        cout << "\n";
    }
    get char( );
    for (i = 0; i < 800; i++)
    {
        * (p+i) = 77;
    }
    for (i = 0; i < 800; i++)
    {
        cout << " " << (p+i);
        cout << " " << * (p+i);
    }
}
```

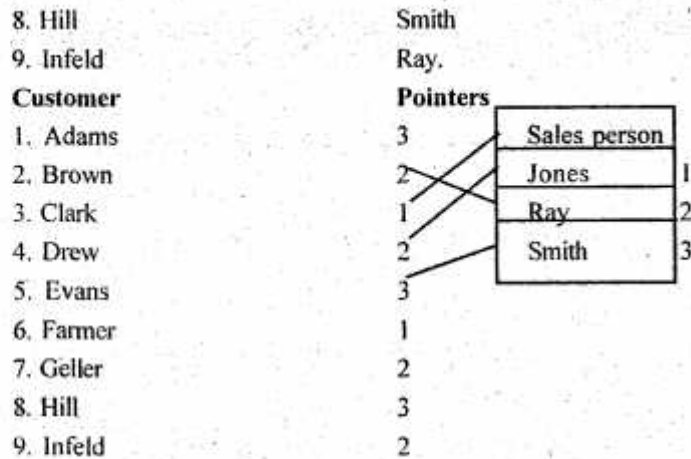
Q. 3. (a) Discuss the advantages and disadvantages of representing a group of items as an array versus a linear linked list.

Ans. Arrays See solution 1(b).

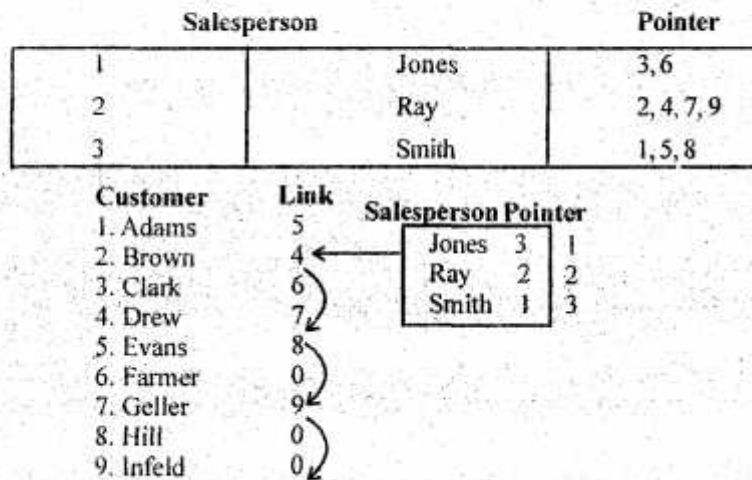
Linked Lists : Linked lists will be introduced by means of an example. Suppose a brokerage firm maintains a file where each record contains a customer's name and his or her sales person, and suppose the file contains the data appearing in figure. Clearly the file could be stored in the computer by such a table, i.e., by two columns of nine names.

However, this may not be the most useful way to store the data, as following discussions shows :

Customer	Salesperson
1. Adams	Smith
2. Brown	Ray
3. Clark	Jones
4. Drew	Ray
5. Evans	Smith
6. Farmer	Jones
7. Geller	Ray



Suppose the firm wants the list of customers for a given salesperson. Using the data representation in figure. The firm would have to search through the entire customer. One way to simplify such a search to have the arrows.



Q. 3. (b) Write algorithm to perform each of the following operations :

- Append an element to the end of a list.
- Reverse a list so that the last element becomes first and so on.
- Combine two ordered list into a single ordered list.

Ans. (i) HNDA (INFO, LINK, START, ITEM, LOC).

This procedure finds the location LOC of the last node in a sorted list such that INFO[loc] < ITEM, or sets.

LOC = NULL

- [List empty?] If START = NULL, then : set LOC := NULL, and return.
- [Special case?] If ITEM < INFO [START], then : set LOC := NULL, and return.
- Set SAVE := START and PTR := LINK [START]. [Initializes pointers]
- Repeat steps 5 and 6 while PTR ≠ NULL.

5. If ITEM < INFO [PTR], then :
Set LOC := SAVE, and Return
[End of if structure].
6. Set SAVE := PTR and PTR := LINK [PTR].
[Updates pointers]
[End of step 4 loop].
7. Set LOC := SAVE.
8. Return.

(ii)

```

q = s = null; /* q is one steps behind p; */
              /* s is two steps behind p. */
for (p = table; p != null && k(p) != key; p = next (p)) {
    s = q;
    q = p;
} /* end for */
if (p == null)
    return (p);
if (q == null)
    return (p);
next (q) = next (p);
next (p) = q;
(s = null)? table = p : (next (s)) = p;
return (p);
    
```

(iii)

Step 1 : [Initialization]

New₁ = stars

Step 2 : Perform insertion

Repeat while New₁ <> NULL

(i) temp = New₁

(ii) New₁ = Next [New₁]

(iii) Found = 0

(iv) Node = Start

Repeat through while Node <> Null & found <> 0.

(v) If Info[Node] = Info[New₁]

(a) Next [temp] = Node

(b) Previous [temp] = previous [node]

(c) Next [Previous [node]] = temp

(d) Previous [Node] = temp

(e) Found = 1 else Node = Next [Node].

(vi) If found <> 0

- (viii) If info [node] > Info temp.
- (a) Next [temp] = Node
 - (b) Previous [temp] = Previous [Node]
 - (c) Next [Previous[Node]] = temp
- else
- (a) Next [temp] = Null
 - (b) Previous [temp] = Node
 - (c) Next [node] = temp

Step 3 : Exit.

Q. 4. (a) What is stack? Explain the functions performed by the stack related operation : push, pop, stacktop and empty.

Ans. A stack is a list of elements in which an element may be inserted or deleted only at one end, called. The top of the stack.

This means, in particular, that elements are removed from a stack in the reverse order of that in which they were inserted into the stack.

Special terminology is used for two basic operations associated with the stacks.

(a) Push

(b) Pop

Push : Used to insert an element into a stack.

Pop : Used to delete an element from a stack.

Stacktop(s) : Which returns the top element of a stack without removing it from the stack.

$x = \text{pop}(s);$

$\text{push}(s, x);$

We represent C function stacktop for a stack of integers as follows :

```
int stacktop (struct stack * ps)
{
    if (empty (ps)) {
        printf("%s", "stacks underflow");
        exit(1);
    }
    else
        return(ps->items[ps->top]);
}
```

Empty Stack : There is no upper limit on the number of items that may be kept in a stack, since the definition does not specify how many times are allowed in the collection. Pushing another item onto a stack merely produces a larger collection of items. However, if a stack contains a single item and stack is popped, the resulting stack contains no items and is called the empty stack.

Q. 4. (b) Transform each of the following expression to prefix and postfix :

(i) $(A + B) * (S(D - E) + F) - G$

(ii) $A + ((B - C) * (D - E) + F) / G(S(H - J))$

Ans. (i) From Infix to Postfix :

Label no.	Symbol Scanned	Stack	Expression
1	((-
2	A	(A
3	+	(+	A
4	B	(+	AB
5)		AB+
6	*	*	AB+
7	(*(AB+
8	\$	*(AB+
9	(*(AB+
10	D	*(AB+D
11	-	*(AB+D
12	E	*(AB+DE
13)	*(AB+DE-
14	+	*(AB+DE-+
15	F	*(AB+DE-+F
16)	*	AB+DE-+F\$
17	-	-	AB+DE-+F\$*
18	G	-	AB+DE-+F\$*G
19			AB+DE-+F\$*G-

(iii) Infix to Postfix :

Lable No.	Symbol Scanned	Stack	Expression
1	A	+	A
2	+	+(A
3	(+(A
4	(+(A
5	B	+(AB
6	-	+(AB
7	(+(ABC
8)	+(ABC-
9	*	+(ABC-
10	(+(ABC-
11	D	+(ABC-D
12	-	+(ABC-D
13	E	+(ABC-DE
14)	+(ABC-DE-

15	+	+(+	ABC-DE-
16	F	+(+/-	ABC-DE-*
17	/	+	ABC-DE-*F
18	G	+\$	ABC-DE-*FG
19)	+\$(-	ABC-DE-*FGI+
20	\$	+	ABC-DE-*FGI+
21	(+\$(-	ABC-DE-*FGI+H
22	H	+\$(-	ABC-DE-*FGI+HJ-
23	-	+\$	ABC-DE-*FGI+HJ-+\$
24	J		
25)		
26			

Q. 4. (c) Write a C program to convert a prefix string to postfix.

Ans. Let Q be an arithmetic expression written in infix notation. Besides operands & operators, Q may also contain left and write parenthesis. We also assume that operators in Q consists only of exponentiations (\uparrow), multiplications (*), divisions (/), additions (+) and subtractions (-) and that they have the usual three levels of precedence.

We also assume that operators on same level including exponentiations, are performed from left to right unless otherwise indicated by parenthesis.

The following algorithm transforms the infix expression Q into its equivalent postfix expression P. The algorithm uses a stack to temporarily hold operators and left parenthesis. The postfix expression P will be constructed from left to right using the operators and left parenthesis.

Algorithm : POLISH(Q, P)

Suppose Q is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression P.

1. Push "(" onto STACK, and add ")" to the end of Q.
2. Scan Q from left to right & repeat steps 3 to 6 for each element of Q until the stack is empty.
3. If an operand is encountered, add it to P.
4. If a left parenthesis is encountered, push it out STACK.
5. If an operator \otimes is encountered, then :
 - (a) Repeatedly pop from STACK and add to P each operator (on the top of STACK) which has the same precedences as or higher precedence than \otimes .
 - (b) Add \otimes to STACK.
6. If a right parenthesis is encountered, then :
 - (a) Repeatedly pop from STACK and add to P each operator until a left parenthesis is encountered
 - (b) Remove the left parenthesis.

[end of if structure]
[end of step 2 loop]
7. Exit.

Q. 5. (a) Write an algorithm to determine the sum of contents of all the nodes in a binary tree.

Ans. FIND (INFO, LEFT, RIGHT, ROOT, ITEM, LOC, PAR).

(i) LOC = NULL and PAR = NULL, will indicate that the tree is empty.

(ii) LOC \neq NULL and PAR = NULL will indicate that ITEM is the root of T.

(iii) LOC = NULL, and PAR \neq NULL will indicate that ITEM is not in T and can be added to T as a child of node N with location PAR.

1. [Tree empty?]
If ROOT = NULL, then : set LOC := NULL and
PAR := NULL and
Return.
2. [ITEM at root?]
If ITEM : INFO [ROOT], then :
Set LOC := NULL and
PAR := NULL and
Return
3. If ITEM < INFO [ROOT], then :
set PTR := LEFT [ROOT] and
Save := Root;
else
Set PTR := RIGHT [Root] and SAVE := Root
4. Repeat step 5 and 6 while PTR \neq NULL;
5. [ITEM found?]
If ITEM = INFO [PTR], then : Set LOC := PTR and PAR := SAVE, and Return.
6. If ITEM < INFO [PTR], then :
set SAVE := PTR and PTR := LEFT [PTR].
else ;
Set SAVE := PTR and PTR := RIGHT [PTR]
[end if structure]
[end of step 4 loop].
7. [Search Successful] set LOC : NULL & PAR := SAVE.
8. Exit.

Q. 5. (b) Write an algorithm to determine if a binary tree is strictly binary.

Ans. Although natural trees grow with their roots in the ground and their leaves in the air, computer scientists almost universally portray tree data structures with the root at the top and leaves at the bottom.

The direction from the root to the leaves is "down" and the opposite direction is "up". Going from the leaves to the root is ("down") called "climbing". The tree, & going from the root to the leaf is called "descending" the tree.

If every non-leaf node in a binary tree has non-empty left & right sub-trees, the tree is termed as strictly binary tree. Thus,

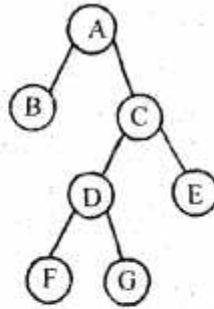


Fig. is strictly binary-tree. A strictly binary tree with n leaves always contain $(2n-1)$ nodes.

Q. 5. (c) Show that a strictly binary tree with n leaves contains $(2n-1)$ nodes.

Ans. The number NUM of nodes in T is 1 more than the number NUML of nodes in the left subtree of T plus the number NUMR of nodes in right subtree of T.

Procedure : COUNT (LEFT, RIGHT, ROOT, NUM).

This procedure finds the number NUM of nodes in a binary tree T in memory.

1. If ROOT = NULL, then : set NUM := 0 and Returns.
2. Call COUNT (LEFT, RIGHT, LEFT [ROOT], NUML).
3. Call COUNT (LEFT, RIGHT, RIGHT [ROOT], NUMR);
4. Set NUM := NUML + NUMR + 1.
5. Return.

Q. 6. (a) Implement the sequential search algorithm for linked lists.

Ans. SEARCH (INFO, LINK, START, ITEM, LOC).

List is a linked list in memory. This algorithm finds the location LOC of the nodes where item first appears in LIST, or sets LOC = NULL.

1. Set PTR := START.
2. Repeat step 3 while PTR \neq NULL
3. If ITEM = INFO [PTR], then:
set LOC := PTR, and exit.
else :
set PTR := LINK [PTR].
[PTR now points to the next node].
[end of if structure].
[end of step 2 loop].
4. [Search is unsuccessful].
set LOC := NULL;
5. Exit.

Q. 6. (b) Giving suitable examples, clearly distinguish between a height-balanced tree and a weight-balanced tree.

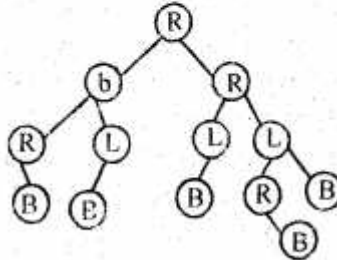
Ans. Balanced binary trees are classified into 2 categories :

(a) Height Balanced.

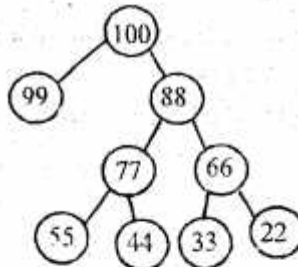
(b) Weight Balanced.

Height Balanced Binary Trees : A tree is called AVL tree, (Balances Binary Tree) if each node possesses one of the following properties :

1. A node is called left heavy if the largest path in its left sub-tree is one longer than the longest path of its right sub-tree.
2. A node is called right heavy if the longer path in the right subtree is one longer than the longest path in its left subtree.
3. A node is called balanced if the longest paths in both the right and left sub-tree are equal. In example the figure illustrates a balanced binary trees.



Balanced binary tree



Unbalanced binary tree

B \Rightarrow Balanced.

L \Rightarrow Left Heavy

R \Rightarrow Right Heavy

A node in a binary tree that does not contain the above properties is said to be unbalanced.

If one inserts a new node into a balanced binary tree at the leaf then the possible changes in the status of a node on the path (balanced indicator), which can occur are as follows :

- * The node was either left or right heavy and has now become balanced.
- * The node was balanced and has now become left or right heavy.
- * The node was heavy and the new node has been inserted in the heavy sub-tree, thus creating an unbalanced subtree, such a node is called a critical node.

First condition to current node is applied, then the balanced indicators of all the ancestor nodes of this node remains unchanged, since the length of the longest path in the sub-tree remains unchanged.

Q. 7. (a) Develop an algorithm for Quick Sort sorting method. Discuss its advantages over other sorting techniques.

Ans. Algorithm : Quick (A, N, BEG, END, LOC).

Here A is an array with N elements, parameters BEG and END contains the boundary values of the sublist of A to which this procedure applies. LOC keeps track of the position of the first element A[BEG] of the sublist during the procedure. The local variables LEFT & RIGHT will contain the boundary values of the list of elements that have not been scanned.

1. [Initilize] set $LEFT := BEG$, $RIGHT := END$ and $LOC := BEG$.
2. [Scan from right to left.]
 - (a) Repeat while $A[LOC] \leq A[RIGHT]$ and $LOC \neq RIGHT$:
 $RIGHT := RIGHT - 1$
[end of loop]
 - (b) If $LOC = RIGHT$, then : Return.
 - (c) If $A[LOC] > A[RIGHT]$, then:
 - (i) [Interchange A [LOC] and A [RIGHT]],
 $TEMP := A[LOC]$, $A[LOC] := A[RIGHT]$,
 $A[RIGHT] := TEMP$.
 - (ii) Set $LOC := RIGHT$
 - (iii) Go to step 3.
3. [Scan from left to right].
 - (a) Repeat while $A[LEFT] \leq A[LOC]$ and $LEFT \neq LOC$:
 $LEFT := LEFT + 1$
[End of loop]
 - (b) If $LOC = LEFT$, then : Return.
 - (c) If $A[LEFT] > A[LOC]$, then
 - (i) Interchange A[LEFT] and A[LOC].
 $TEMP := A[LOC]$, $A[LOC] := A[LEFT]$,
 $A[LEFT] := TEMP$.
 - (ii) Set $LOC := LEFT$.
 - (iii) go to step 2

Advantages :

- (i) It is possible to speed up quick sort for sorted files by choosing a random element of each sub-file as the pivot value. If a file is known to be nearly sorted, this might be a good strategy.
- (ii) The analysis for the case in which the file size is not an integer power of 2 is similar but slightly more complex; the results, however, remain the same.
- (iii) Mean sort is $O(n \log n)$ as long as the elements of the file are uniformly distributed between larger and smallest.
- (iv) B sort requires for less time than quicksort or mean sort on sorted or nearly sorted input, although it does require more comparisons and interchanges than meansort for nearly sorted input.

- (v) The space requirements for the quicksort depend on the number of nested recursive calls or on the size of the stack.
- (vi) Another advantage of quicksort is locality of reference. That is, over a short period of time all array accesses are one to one in two relatively small portions of the array. This ensures efficiency in the virtual memory environment, where pages of data are constantly being swapped back & forth between external & internal storage.

Q. 7. (b) Write a detailed note on the application of graphs.

Ans. (i) C Representation of Graphs : Suppose that the number of nodes, in the groups is constant; that is, arcs may be added or deleted but nodes may not. A graph with 50 nodes could then be declared as :

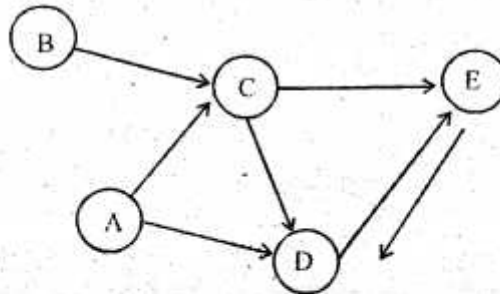
```
#define MAX_NODES 50.
struct node {
};
struct arc {
int adj;
};
struct graph {
struct node nodes [MAX_NODES];
struct arc arcs [MAX_NODES] [MAX_NODES];
};
```

Frequently, the nodes of a graph are numbered from 0 to MAX_NODES-1 and no information is assigned to them. Also, we may be interested in the existence of arcs but not in any weights & other information about them :

(ii) Transitive Closure : Let us assume that a graph is completely described by its adjacency matrix, adj. Consider the logical expression $\text{adj}[i][k] \&\& \text{adj}[k][j]$.

Now, consider the expression :

$$(\text{adj}[1][0] \&\& \text{adj}[0][j]) \vee (\text{adj}[1][j] \&\& \text{adj}[j][j])$$



	A	B	C	D	E
A	0	0	1	1	0
B	0	0	1	0	0
C	0	0	0	1	1
D	0	0	0	0	1

	A	B	C	D	E
A	0	0	0	1	1
B	0	0	0	1	1
C	0	0	0	1	1
D	0	0	0	1	0

E 0 0 0 1 0
(a) adj 1

E 0 0 0 0 1
(b) adj 2

(iii) **Shorter Path Algorithm** : In a weighted graph, or N/w, it is frequently desired to find the shortest path between the nodes, S and t. The shortest path is defined as a path from S to t such that the sum of the weights of the arcs on the path minimized.

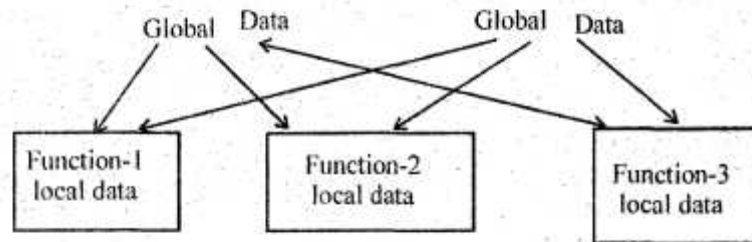
Q. 8. Write explanatory notes on any two of the following :

(i) Top-down design

(ii) Merge Sort

(iii) Doubly linked lists

Ans. (i) Top-down Design :



Relationship of data & functions in procedural programming

Some characteristics exhibited by procedure, oriented programing emphasis is on doing things (algorithms).

Large programs are divided into smaller programs known as functions.

Most of functions share global data.

Data move openly around the system from function to function.

Functions transform data from one to another form.

Employs top-down approach in program design.

Global data are more unversable to an inadvestent change by a function.

(ii) **Merge Sort** : Suppose an array A with n elements $A[1], A[2], \dots, A[N]$ is in memory. The merge sort algorithm which sorts A will first be described by means of a specific example.

Suppose the array A contains 14 elements as follows :

60, 33, 40, 22, 55, 88, 60, 11, 80, 20, 50, 44, 77, 30.

Each pass of the merge-sort algorithm will start at the begining of the array A now merge pairs of sorted subarrays as follows :

Pass 1 : Merge each pair of elements to obtain the following list of sorted parts

33, 66 22, 40 55, 88 11, 60 20, 80 44, 50 30, 77

Pass 2 : Merge each pair of pairs to obtain the following list of sorted quadraples.

22, 33, 40, 66 11, 55, 60, 88 20, 44, 50, 80 30, 77

Pass 3 :

11, 20, 22, 33, 40, 44, 50, 55, 60, 66, 77, 80, 88

Pass 4 :

11, 22, 33, 40, 55, 60, 66, 88 20, 30, 44, 50, 77, 80

The above merge sort algorithm for sorting an array A has the following important property :

After pass k, the array A will be partitioned into sorted sub-arrays where each sub-array, except possibly the last, will contain exactly $L = 2^k$ elements.

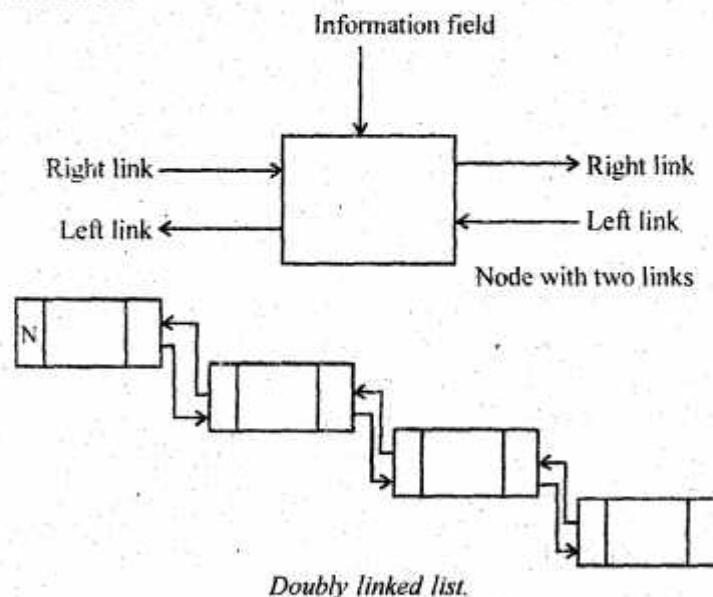
Hence the algorithm requires at most $\log n$ passes to sort an n-element array A.

The merge pass procedure applies to an n-element array A which consists of a sequence of sorted sub-arrays.

Moreover, each subarray consists of L elements except that the last subarray may have fewer than L elements. Dividing n by $L \cdot L$, we obtain the quotient Q, which tells the number of pairs of L-elements sorted subarrays, that S'.

$$Q = \text{INT}(N / (2 * L)).$$

(iii) Doubly Linked Lists :



A linear linked list structure traversing is possible only in one direction. Sometimes it is required to traverse the list in either direction, forward or backward.

This increases the performance and efficiency of algorithms, so traversing of linked list in both the directions requires as illustrated in figure.

The links are used to denote the predecessor and successor its right link. A list with this type of arrangement is called doubly linked list. Now it is possible to define a doubly linked lists as a collection of nodes, each node having three fields :

- (i) Pointer to previous node (Pointer to predecessor).
- (ii) Information field.

(iii) Pointer to next node (Pointer to successor).

How one can create a node of doubly linked, is given below :

Step 1 : [Initialization]

- (i) Start.Next = NULL
- (ii) Previous.Next = NULL

Step 2 : [Assign address of start variable to node].

Node = Address of start.

Step 3 : [Make left and right links of a node].

- (i) Next [Node] = Node [Make link to node].
- (ii) Previous [Node] = Node [Make link to node].
- (iii) Node = Next [Node] [Move pointer to next node].
- (iv) Info[Node] = Value [Information value of a node].
- (v) Next [Node] = Null [Assign NULL to the address field].

Step 4 : [Call step 3]

Repeat step 3 to create required number of nodes for linked list.

Step 5 : End.


```
(ii) #include<stdio.h>
#include<stdlib.h>
void bubble_sort (int_array[ ],int size)
{
    int temp,i,j;
    for (i=0; i<size, i++)
    for (j=0; j<size-1; j++)
    if array [j] <array[j+1]);
    {
        temp = array[j];
        array[j+1]=array[j];
        array[j]=temp,
    }
}
```

Q. 1. (b) What do you understand by Dynamic Memory Management ?

Ans. If the size of file is not known & we have to load it into memory and defining very large array, so that file can fit into the array. If the size of the array is larger than that of file then the space is wasted. If the size of file is larger then it is not possible to load the complete file into the array, leading to a need for dynamic memory management Here, we use a pointer variable and it's size is allocated during the execution of the program.

Q. 2. (a) Write a method to convert an infix expression to postfix notation. Show these steps to convert the following expression to postfix form :

$$(3**2*5)/(3*2-3)+5$$

Ans.	Infix	Postfix
	A + B	AB+
	A + B - C	AB+C-
	(A + B) * (C - D)	AB+CD-*
	A\$B* C-D+E/ F(G+H)	AB\$C*D-EF/GH+/+
	((A + B) * C - (D - E)) \$(F + G)	AB+C*DE--FG+\$
	A - B / (C + D\$E)	ABCDE\$*/-

Q. 2. (b) Write down the algorithm to implement two stacks using only one array.

```
Ans. #include<stdio.h>
#include(string.h)
#include<define.h>
int top = -1;
int flag=0;
int stack [100];
int data;
void push (int *, int);
```