CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)

An Autonomous Institution Accredited with 'A' Grade by NAAC

Department of Computer Science & Electronics

Subject: Data Structures using C

Class: BCA III Semester & B.Sc.(Comp.Sc.) II Year

Prepared By: Prof. N. Jaiswal

Chapter-1 Fundamental Notations

Very Short Answer Questions

Q.1

(i) What is data type?
Ans. A *data type* is a term, which is used to refer the kinds of data that variables may hold in a programming language. The general form of a class of data items is also known as data type.

(ii) Define information.
Ans. Information is a processed data and data is a collection of values (raw data) from which no conclusion is made.

(iii) Define data structure.
Ans. A data structure is defined as a specific way of arranging data and a set of operations performed on the data that make it behave in a certain way.

(iv) Write the name of two built-in data structure. Ans.
Two built-in data structure are int (integer) and float.

(v) What do you understand by linear and non-linear data structure?
Ans. The data structure in which traversing is done in a linear fashion, one by one, is known as linear data structures. On the other hand, in a non-linear data structure, the traversing is not done in a linear fashion.

(vi) List the four major operations performed on linear data structure.
Ans. There are four basic operations that are performed on data structures – traversing, inserting, deleting and searching**.**

(vii) Define algorithm.
Ans. An algorithm is a finite sequence of steps which, if followed, itself perform a well defined task.

(viii) Define program testing.
Ans. Program testing is a process in which a program is validated on all possible sample data. Testing is said to be complete when all desired verification against specification are found valid.

(ix) Define structured programming.
Ans. Structured programming is a computer programming technique in which the statements are organized in a specific manner to minimize error or misinterpretation.

(x) Define debugging.
Ans. *Program debugging* is the process of isolating and correcting the errors. Debugging is finished when there is no errors of any type.

(xi) Define documentation.

Ans. *Documentation* is the written text and comments that makes a program easier for the readers to understand, use and modify.

(xii) Define variable.
Ans. An entity that may change during the execution of a program is called as a variable.

(xiii) What is a constant?
Ans. A constant is an entity that remains fixed throughout the execution of a program. For example – 25, 165.5, 'M', "Hello".

(xiv) Define array.
Ans. An array is a collection of similar type of elements which are stored in memory in contiguous locations.

(xv) Define pointer.
Ans. An address of a data item is known as a pointer.

(xvi) Define pointer variable.
Ans. A variable that hold an address of a data item is called as a pointer variable.

(xvi) What is a structure?
Ans. A structure is a collection of heterogeneous (dissimilar) elements which are stored in memory contiguously.

(xvii) Define global variable.
Ans. Variables which are declared outside all functions are called as global variables. Global variables are accessible to all functions.

(xviii) When the "Overflow" condition comes in data structure?
Ans. When we perform an insertion operation on a data structure and there is not sufficient memory to accommodate it, it is called an "overflow" condition.

(xix) Write a statement that displays the address of an integer variable.
Ans. By using an address of '&' operator with an integer variable, we can display the address of an integer variable. For example:

```
printf("%u", &p);
```

This statement displays the address of 'p' integer variable:

(xx) Scope of a variable
Ans. The portion of a program where a variable can be accessed is called as the scope of that variable. The scope of the variable is determined by its place of declaration.

(xxi) Real numbers
Ans. Numbers with fractional part are known as real numbers. For example, 12.5, 34.725, etc.

(xxii) Functions
Ans. Functions are self-code of statements which are written for performing a specific given task. Each function performs a different function.

(xxiii) Parameters
Ans. Parameters are those entities that are used in function call as well as in function definition. When we call a function we pass arguments to the called function. Such arguments are called actual parameters. And calling function receives these actual arguments in formal parameter (arguments).

Short answer type questions

Q.2 Explain Program Development Life Cycle.
Ans. A typical program development life cycle has following steps:

1. Planning
2. Systems Analysis
3. Systems Design
4. Development
5. Testing
6. Implementation and
7. Maintenance.

Q.3 Explain various types of data type in detail.
Ans. Data can be either primary data type or secondary data type.

Primary data type - The data types, which are not composed of other data types, are called as *primary data type*. C names primary data types as standard data type. Primary data types are integers, floats and characters data types. The primary data types could be of several types. For example an integer could be a short integer or a ling integer. Or a character could be an unsigned char or a signed char.

Secondary data type - The data types, which are composed of primary data types, are called as *secondary data type.* Normally these are defined by the programmer that's why these are sometimes called user – defined data types. In C secondary data types are arrays, structures, pointers, unions, enum etc.

Q.4 What is data structure? Give difference between the data and information.
Ans. A data structure is defined as a specific way of arranging data and a set of operations performed on the data that make it behave in a certain way. Information is a processed data and data is a collection of values (raw data) from which no conclusion is made.

Q.5 Give difference between linear and non-linear data structures.
Ans. The data structures whose components are processed sequentially, one by one, are referred as linear data structures. The data structures whose components are processed randomly are referred as linear data structures.

Q.6 Differentiate between primitive and non-primitive data structure.
Ans. The primitive data structures are those data structures, which are internally provided by the high – level languages, such as Pascal, Fortran, C, C++ etc. However the non-primitive data structures are made from primitive data structures according to the needs of a programmer. These data structures include array, structure, stack, queues, linked list, etc.

Another difference is that there are predefined set of operations on primitive data structures, whereas on non-primitive data structure the user has to define the set of operations exclusively.

Q.7 Explain various type of constants in detail.
Ans. A constant is a number, character, or string literal that cannot be changed during the execution of a program.. C provides four types of constants - Integer constants, Real (floating point) constants, Character constants and String constants

Integer Constants –	1234	-420	100
Real Constants –	**Decimal form**	4312.56	0.5
	Exponential form	12.876e4	-0.2e-5
Character Constants –	'A'	' '	'+'

String Constants – "Lord Krishna is the most powerful warrior of the great epic - Mahabharata."

Q.8 Give the advantages of modular programming.
Ans. The advantages of using modular programming are as:

(i) Easy to debug
(ii) algorithm is more easily understood
(iii) programmers can use their expertise on particular techniques
(iv) testing can be more thorough on each of the modules
(v) allows library programs to be inserted

Q.9 What is pointer variable? Give the advantages of pointer variable.
Ans. A variable that stores the address of an item is known as a pointer variable. Here are some advantages of pointer variable:

(i) Pointers allow us to use dynamic memory allocation.
(ii) Pointers obviously give us the ability to implement complex data structures like linked lists, trees, etc
(iii) Pointers allow ease of programming, especially when dealing with strings.

Q.10 Explain structured programming in detail.
Ans. Structured programming is a programming paradigm aimed on improving the clarity, quality, and development time of a computer program by making extensive use of subroutines, block structures and for and while loops - in contrast to using simple tests and jumps such as the goto statement which could lead to "spaghetti code" which is both difficult to follow and to maintain.

Q.11 What is function? Explain global and local variables used in functions.
Ans. A function is designed to perform a well-defined task. C supports two types of functions - Standard (library) functions and User-defined functions. The functions, which are already defined in the C libraries for the users, are called as standard functions. The standard functions are also called as *library* functions or *readymade* functions. However the functions, which are defined by the user for its sake of convenience, are known as *user-defined* functions.

Global and local variables – The formal arguments and variables declared inside a function block are *local* variables. These *local* variables are not seen by other function in the program. This type of local variable is called as automatic local variable. On the other hand, if the variables are declared anywhere outside all the functions, they can be accessible from any function in the program file. Such variables are called as *global variables*. Global variables are usually declared in the beginning of a program.

Q.12 Write a program to swap the contents of two variables using pointers.
Ans.

```
#include <stdio.h>
main()
{
        int a, b;
        printf("\nEnter any two number –");
        scanf("%d %d", &a, &b);
        printf("\nTwo numbers before calling swap() function - %d %d", a,
        b); swap(&a, &b);
        printf("\nTwo numbers after calling swap() function - %d %d", a, b);
}
swap(int *x, int *y)
{
        int t;
        t = (*x);
```

```
        (*x) = (*y);
        (*y) = t;
}
```

Long Answer Type questions

Q.13 Explain program development life cycle.

Ans. Programming can be defined as the development of a solution to an identified problem. There are seven basic steps in the development of a program:

1. Define the problem - This step (often overlooked) involves the careful reading and re-reading of the problem until the programmer understands completely what is required.

2. Outline the solution (analysis) - Once the problem has been defined, the programmer may decide to break the problem up into smaller tasks or steps, and several solutions may be considered. The solution outline often takes the shape of a hierarchy or structure chart.

3. Develop the outline into an algorithm (design) - Using the solution outline developed in step 2, the programmer then expands this into a set of precise steps (algorithm) that describe exactly the tasks to be performed and the order in which they are to be carried out. This step can use both structured programming techniques and pseudocode (covered in more detail later).

4. Test the algorithm for correctness (desk check) - This step is one of the most important in the development of a program as is often forgotten. Test data needs to be walked though each step in the algorithm to check that the instructions described in the algorithm will actually do what they are supposed to. If logic errors are discovered then they can be easily corrected.

5. Code the algorithm into a specific programming language (coding) - It is only after all design considerations have been met that a programmer should actually start to code the program. In preceding analysis it may have been necessary to consider which language should be used, as each has its own peculiarities (advantages and disadvantages).

6. Run the program on the computer (testing) - This step uses a program compiler and test data to test the code for both syntax and logic errors. If the program is well designed then the usual time-wasting frustration and despair often associated with program testing are reduced to a minimum. This step will often need to be done several times until the programmer is satisfied that the program is running as required.

7. Document and maintain the program (documentation) - Program documentation should not be just listed as the last step in the development process, as it is an ongoing task from the initial definition of the problem to the final test results. Documentation also involves maintenance - the changes that are made to a program, often by another programmer, during the life of that program.

The better a program has been documented and the logic understood, the easier it is for another to make changes. The whole process of defining problems to providing the coded solution are an ongoing process that is circular in nature and can be called the ***System Development Life Cycle*** (SDLC).

Q.14 What are linear and non-linear data structures? Explain with example.

Ans**. Linear data structures -** The data structures whose components are processed sequentially, one by one, are referred as linear data structures. Generally we have following types of linear data structures:

1. Stack - A stack is a data structure in which insertions and deletions are made at one end, called top of the stack. A stack is called as LIFO (Last–In First –Out) structure. The plate holder in a cafeteria has this property. We can take only the top plate. When we do this, the only below it rises to the top so that the next person can take one.

2. Queue - A queue is a data structure in which insertions are made at one end and deletions are made at other end. A queue is also called as FIFO (First–In First–Out) structure. A waiting line in a bank or for a bus is suitable example of a queue.

3. Linked List - A linked list is a data structure in which the components are logically ordered by their pointer fields rather than physically ordered. Each component of the linked list, called a node, has two fields – data field and link field. The data field stores the information and the link field points to the next component.

Non–linear data structures – In non–linear data structures, elements are not processed linearly. In these insertion and deletion can take place anywhere in the data structure. Non–linear data structures are of two main types:

1. Tree – Tree is a non–linear data structure and is defined as a finite set of one or more nodes such that

(i) there is a specially designated node, called the root node of the tree

(ii) the remaining nodes are partitioned into $n \geq 0$ disjoint sets $T_1, T_2, \ldots, T_n$ are called the subtree of root node

Topmost node of a tree is called the root of the tree and remaining nodes are called leaves (children) of the tree.

2. Graph – Graph is another important non-linear data structure, which is represented by a 2-tuple (V,E) such that V is a set of vertices and E is a set of edges. We write it as:

$G = (V, E)$ where $V = (v_1, v_2, \ldots v_n)$ and $E = (e_1, e_2, \ldots e_n)$

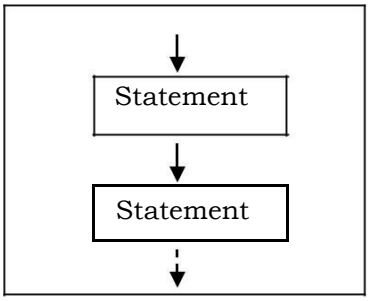
Here 'E' is a set of pairs of vertices and these pairs are called edges.

Q.15 What is structured programming? Explain top- down and bottom-up design for solving any problem. Illustrates with examples.

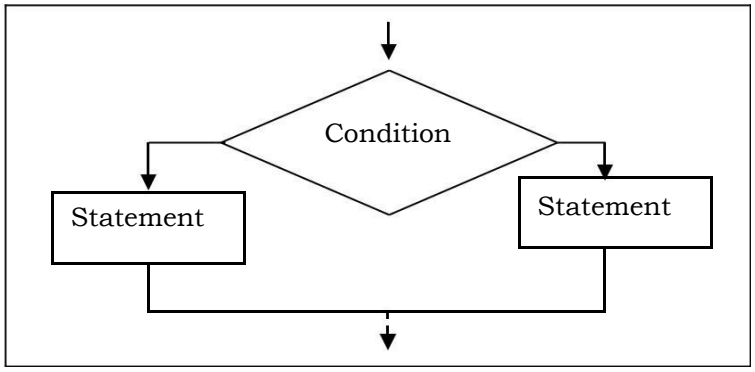
Ans. Structured programming is a computer programming technique in which the statements are organized in a specific manner to minimize error or misinterpretation. Structured programming is an organized programming approach that involve the use of four basic control structures, **- Sequential, Conditional, Repetition** and **Procedures**.

The **sequential** structure is composed of statements executed one after another. In this all the statements are executed in the order in which they are written. The **conditional** structure (also know as selection structure) executes different set of statements depending upon certain condition. The **repetitive** structures repeat a set of statements while certain conditions are met. The **procedure** enables us to replace a set of statements with a single statement. Following figure shows these basic structures of programming languages.

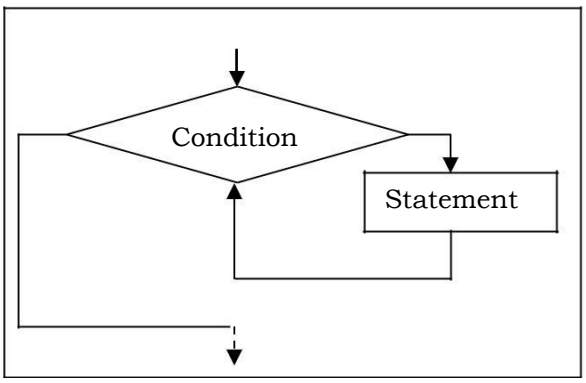
Sequence



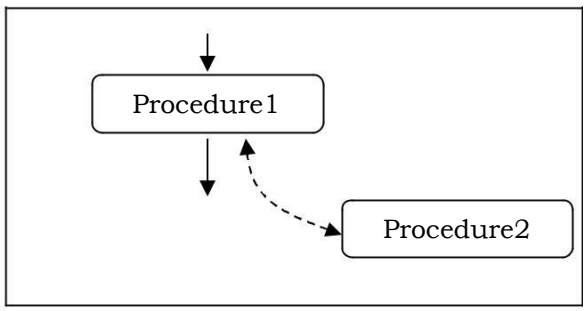
Selection



Repetition (Loop)



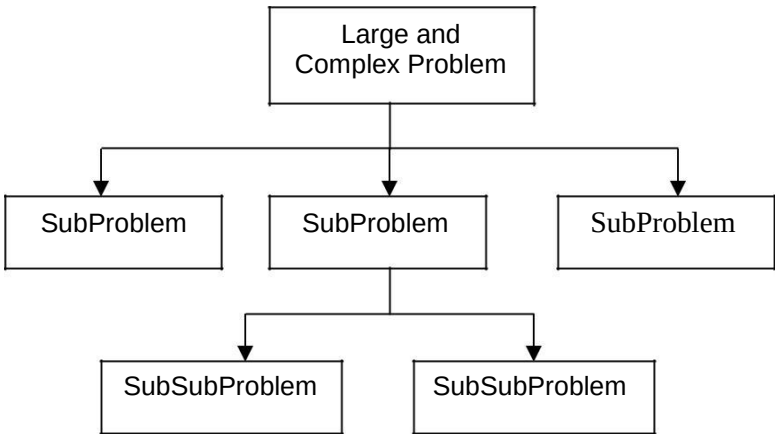
Procedures (Functions)



A procedure is a combination of the basic structures that is considered as a single statement in the program. In C language procedures are called as *functions*. They allow you to write parts of your programs separately, then assemble into final form. In other words you can say that structured programming approach uses the top-down approach to decompose main functions into lower level components for modular coding process. This technique improves the programming process through better organization of programs and better programming notations to facilitates, correct, and clear description of data in control structures.

To handle large and complex problem we must follow top-down approach.

Top-down and Bottom-up design – The **top-down** approach divides the large problem into smaller problems (subproblems) that can be handled more easily. However if the subproblem is still complex then it must be further divided into subsubproblems. This process goes on until each subproblem can not be further divided. Each subproblem can be easily solved independently of the others. These subproblems are also called as modules. That's why the top-down approach is also called as modular programming. Following figure shows top-down methodology.



The top-down approach can be broken down as:

1. **Problem Definition -** Understand the problem statement, that is one must very clear about what is given (input) and what is required (output)
2. **Naming the Modules and Submodules –** Write the naming module and name the lower level modules
3. **Algorithms and their Implementations -** Write the remaining submodules
4. **Analyze –** Resequence and revise, if necessary

Another interesting approach is **bottom-up** approach. This approach works inversely of top-down. In bottom-up approach the programmer might choose to solve different parts of the problem directly in his programming language and then combine these pieces into a complete program. Experience suggests that the top-down approach should be followed when creating a program.

Chapter-2 Arrays

Very Short Answer Type Questions

Q.1

(i) What do you mean by an array?
Ans. An array is a collection of similar data items which are stored in memory contiguously.

(ii) Declaration of single dimensional array. Ans. An array is declard as:

datatype arrayname[Size];

Here arrayname is the name of array, Size is the number of elements in the array and datatype specifies the type of elements stored in array.

(iii) What do you understand by index or subscript of an array?
Ans. The index or subscript of an array is used to access the individual element of the array.

(iv) What is base address of array?
Ans. The address of the very first element of an array is known as the base address of an array.

(v) Give formula to calculate the size of one dimensional array.
Ans. The size of one dimensional array = UB – LB + 1. Here UB and LB are upper and lower boundary of one dimensional array.

(vi) If UB is 15 and LB is -7, find the length of array.
Ans. Here UB is 15 and LB is -7, therefore the length of array = (15- (-7)+1)
= (15+7+1)
= 23

(vii) Given a linear array A(5:50) B(18). Find the number of elements in each array. Ans. The length of array A = (50-5+1)
= 46

The length of array B = (18-0+1)
= 19

(viii) Define two dimensional arrays.
Ans. A two-dimensional array is basically a collection of similar elements in which element are accessed by two subscript.

(ix) What is row major order?
Ans. Row major order is a storage allocation scheme for two-dimensional array in which the elements of two-dimensional array are stored row by row, that is first row of two-dimensional array is stored first, then second, third, fourth.

(x) Write formula to calculate length of two dimensional arrays.
Ans. The two dimensional array is represented either in row major order or in column order.

Row major Order - The basic formula for calculating the address of element A[I, J] in row major order of an array of (M x N) elements is as:

A[I,J] (Address of [I, J]) = Base + Size (N x (I - M1) + (J - N1))

Here Base is the base address of a two dimensional array, Size specifies the size of element in bytes, N is total number of columns (N2 - N1+ 1), N1 is starting column number, N2 is last column number and M1 is first row number.

Column major Order - The basic formula for calculating the address of element A[I, J] in an array of (M x N) elements in column major order as :

A[I,J] (Address of [I, J]) = Base + Size (M x (I – N1) + (J – M1))

Here M is total number of columns (M2 - M1+1), M1 is starting row number, M2 is last row number and N1 is starting column number.

(xi) What do you mean by multidimensional array?
Ans. Arrays having more than one dimension are known as multidimensional arrays. Two dimensional arrays and three dimensional arrays are example of multidimensional arrays.

(xii) How does a structure differ from an array?
Ans. A structure is a collection of heterogeneous (dissimilar) elements which are stored in memory contiguously, whereas an array is a collection of homogeneous (similar) elements which are stored in memory contiguously

(xiii) Which built-in functions are used to allocate memory dynamically?
Ans. The calloc() and malloc() functions are used to allocate memory dynamically.

Short Answer Type Questions

Q.2 What is linear array? How is it represented in memory?
Ans. A linear array is a collection of similar data items whose individual item can be accessed by using an index that indicates the position of the item within the collection.

Memory representation of a linear array – The elements of a linear array are stored in contiguous memory locations. It means that if there is an integer array of 10 elements starting from the address 6000, then the next element would be stored at address 6002, and the third element would be stored at 6004, and so on. Figure shows this:

Address 6000 6002 6006 6018

Q.3 Consider the one dimensional float array LA. Base address is 2000, w = 4, LB = 1 and UB = 8. Calculate the address of LA[4] and LA[6].

Ans. The formula Address of finding the address of Ith data is as:

LA[I] = BaseAddress + w * (I – LB)

Thus address of LA[4] = 2000 + 4 * (4 – 1) = 2000 + 4 * 3 = 2000 + 12
= 2012
Address of LA[6] = 2000 + 4 * (6 – 1) = 2000 + 4 * 5 = 2000 + 20
= 2020

Q.4 Write the steps for traversing a linear array.
Ans. In traversing each and every element of array is traversed exactly once. Here is the traversing algorithm.

Traverse (A, N)
Here A is the base address of array, N is the size of an array

1. Set I = 0
2. Repeat through step-4 while (I< N)
3. Process element A[I]
4. Increment I as I = I+1
5. Exit

Q.5 Write an algorithm for search operation of an array.
Ans. **Search (A, N, Item)**
Here **A** is the base address of array, **N** contains total number of elements in an array and **Item** be the data item to be searched

1. Reset Flag = 0
2. Initialize I = 0
3. Repeat through step-6 while (I< N)
4. If Item = A[I] then go to step 5; otherwise go to step 6
5. Set Flag = 1 and go to step-7
6. Increment the value of I as I = I+1
7. If Flag = 1 then print the message – "Item found"; otherwise print the message – "Item not found"
8. Exit

Q.6 Write a program which defines the concept of traversing of an array.
Ans.

```
main()
{
        int i;
        int a[10];
        printf("\nEnter any 10 array elements.\n");

        Traverse(a, 10);  /* Traversing all array elements */
}
Traverse(int a[], int n)
{
        int i;
        for (i = 0; i<n; i++)
                printf("\n%d", a[i]);
}
```

Long Answer Type questions

Q.7 Explain how two dimensional array is represented in memory.
Ans. Basically there are two types of storage allocation for two-dimensional array:

(a) Row major order
(b) Column major order

Row major order - In this the elements of two-dimensional array are stored row by row, that is first row of two-dimensional array is stored first, then second, third, fourth. And so on. Let we have a matrix of (3x4) dimension:

$$\begin{pmatrix} 4 & 9 & 7 & 8 \\ 2 & 5 & 0 & 8 \\ 1 & 6 & 1 & 3 \end{pmatrix}_{3x4}$$

The row major order of this above matrix is as:

4	9	7	8	2	5	0	8	1	6	1	3

1st Row 2nd Row 3rd Row

The computer stored the base address of the array and using this base address we can find out the address of other elements. The basic formula for calculating the address of element A[I, J] in an array of (M x N) elements is as:

A[I,J] (Address of [I, J]) = Base + Size (N x (I - M1) + (J - N1))

Here Base is the base address of a two dimensional array, Size specifies the size of element in bytes, N is total number of columns (N2 - N1+ 1), N1 is starting column number, N2 is last column number and M1 is first row number.

Column major order - In this the elements of two-dimensional array are stored column by column, that is first column of two-dimensional array is stored first, then second, third, fourth. And so on. Let we have a matrix of (3x4) dimension:

$$\begin{pmatrix} 4 & 9 & 7 & 8 \\ 2 & 5 & 0 & 8 \\ 1 & 6 & 1 & 3 \end{pmatrix}_{3x4}$$

The row major order of this above matrix is as:

4	2	1	9	5	6	7	0	1	8	8	3

1st Col 2nd Col 3rd Col 4th Col

The basic formula for calculating the address of element A[I, J] in an array of (M x N) elements in column order as :

A[I,J] (Address of [I, J]) = Base + Size (M x (I – N1) + (J – M1))

Here M is total number of columns (M2 - M1+1), M1 is starting row number, M2 is last row number and N1 is starting column number.

Q.8 Explain the various operations associated with two dimensional arrays.
Ans. The matrices are excellent example of two dimensional arrays. There are following major operations associated with matrices:

Addition of Matrices – Two matrices are added only when their dimensions are exactly same. The algorithm of performing addition operation on matrixes A & B is as:

AddMmatrix (A, Row1, Col1, B, Row2, Col2, C)
Here 'A' and 'B' matrices are of sizes (Row1 x Col1) and (Row2 x Col2) respectively. And C is the resultant matrix.

1. If ((row1!= row2) Or (col1!=col2)) then go to step 9
   otherwise go to step 2
2. Initialize I = 0
3. Repeat through step-8 while (I < Row1)
4. Initialize J = 0
5. Repeat through step-7 while (J < Col1)
6. Add A[I][J] and B[I][J] and store the result in C=I][J]
   as: C[I][J] = A[I][J] + B[I][J]
7. Increment J as J = J + 1
8. Increment I as I = I +1
9. Exit

Multiplication of Matrices – Two matrices are multiplied only when number of columsn of first matrix is equal to number of rows of first second matrix. The algorithm of performing multiplication operation on matrixes A & B is as:

MultMmatrix (A, Row1, Col1, B, Row2, Col2, C)
Here 'A' and 'B' matrices are of sizes (Row1 x Col1) and (Row2 x Col2) respectively. And C is the resultant matrix.

1. If (Col1!= Row2) then go to step 13; Otherwise go to step 2
2. Initialize I = 0
3. Repeat through step-12 while (I < Row1)
4. Initialize J = 0
5. Repeat through step-11 while (J < Col2)
6. Set C[I][J] = 0
7. Initialize k = 0
8. Repeat through step-10 while (J < Col1)
9. Compute C[I][J] = C[I][J] + A[I][K] * B[K][J]
10. Increment K as K = K + 1
11. Increment J as J = J + 1
12. Increment I as I = I +1
13. Exit

Transpose of a Matrix - Let we have a matrix A of size (m * n). The transpose of A, written as A^T, is obtained by interchanging the rows with corresponding columns of a matrix A. The ij^{th} element of B is given by

$$B_{ij} = A_{ji}$$

The algorithm of performing transpose of a given on matrix A is as:

Transpose (A, Row1, Coll1, B)
Here A is a matrix of having Row1 and Col1 number of rows and number of columns respectively.

1. Initialize I = 0
2. Repeat through step-7 while (I < Row1)
3. Initialize J = 0
4. Repeat through step-6 while (J < Col1)
5. Assign B[I][J] = A[J]I]
6. Increment of J as J = J + 1
7. Increment of I as I = I +1
8. Exit

Q.9 Write a program in 'C' to add two matrices.
Ans. Here is the program in 'C' to add two matrices:

```
#include <stdio.h>
main()
{
    int a[10][10], b[10][10], c[10][10];
    int i, j, row1, row2, col1, col2;

    printf("\nEnter the row and column of first matrix :
    "); scanf("%d %d", &row1, &col1);

    printf("\nEnter the row and column of second matrix : ");
    scanf("%d %d", &row2, &col2);

    if ((row1 == row2) && (col1 == col2))
    {
        printf("\nReading the contents of first matrix (%d*%d).\n", row1, col1);
        for(i=0; i<row1; i++)
        {
            for(j=0; j<col1; j++)
            {
                scanf("%d", &a[i][j]);
            }
        }
        printf("\nReading the contents of second matrix (%d*%d).\n", row2, col2);
        for(i=0; i<row2; i++)
        {
            for(j=0; j<col2; j++)
            {
                scanf("%d", &b[i][j]);
            }
        }
        /* Adding two matrices */
        for(i=0; i<m; i++)
```

```
            {
                for(j=0; j<n; j++)
                    c[i][j] = a[i][j] + b[i][j];
            }

            printf("\nAddition of two matrices is as:\n");
            for(i=0; i<row1; i++)
            {
                printf("\n");
                for(j=0; j<col1; j++)
                {
                    printf("%d\t", c[i][j]);
                }
            }
        }
        else
            printf("\nYou have entered wrong dimensions.\n");
    }
```

Q.10 Write a program in 'C' to multiply two matrices.
Ans. Here is the program in 'C' to multiply two matrices:

```
    #include <stdio.h>
    main()
    {
        int a[3][3], b[3][3], c[3][3];
        int i, j, k, r1, r2, c1, c2;

        printf("\nEnter the elements of first matrix : ");
        for(i=0; i<3; i++)
        {
            for(j=0; j<3; j++)
                scanf("%d", &a[i][j]);
        }
        printf("\nEnter the elements of second matrix : ");
        for(i=0; i<3; i++)
        {
            for(j=0; j<3; j++)
                scanf("%d", &b[i][j]);
        }
        /* Multiplying two matrices */
        for(i=0; i<3; i++)
        {
            for(j=0; j<3; j++)
            {
                c[i][j] = 0;
                for(k=0; k<3; k++)
                    c[i][j] = c[i][j] + a[i][k] * b[k][j];
            }
        }
        printf("\nMultiplication of two matrices is as:\n");
        for(i=0; i<3; i++)
        {
            printf("\n");
            for(j=0; j<3; j++)
```

```
                printf("%d\t", c[i][j]);
        }
    }
```

Q. 11 Write a program in C to add two matrices using pointers.

Ans. A program in C to multiply two matrices using pointers.

```
#include <stdio.h>
#include <alloc.h>
main()
{
        int i, j, r1, c1, r2, c2, *a, *b, *c;
        printf("\nEnter number of rows and columns of first matrix -
        "); scanf("%d %d", &r1, &c1);
        printf("\nEnter number of rows and columns of second matrix - ");
        scanf("%d %d", &r2, &c2);
        if ((r1 == r2) && (c1 == c2))
        {
                a = (int *)malloc(r1*c1*2);
                b = (int *)malloc(r2*c2*2);
                c = (int *)malloc(r1*c1*2);
                printf("\nEnter elements of first matrix - ");
                for(i=0; i<r1; i++)
                {
                        for(j=0; j<c1; j++)
                                scanf("%d", &a[i*r1+j]);
                }
                printf("\nEnter elements of first matrix - ");
                for(i=0; i<r1; i++)
                {
                        for(j=0; j<c1; j++)
                                scanf("%d", &b[i*r1+j]);
                }
                printf("\nAddition of two matrices is \n");
                for(i=0; i<r1; i++)
                {
                        for(j=0; j<c1; j++)
                          c[i*r1+j] = a[i*r1+j] + b[i*r1+j];
                }
                for(i=0; i<r1; i++)
                {
                        printf("\n");
                        for(j=0; j<c1; j++)
                                printf("%d\t", c[i*r1+j]);
                }
        }
        else
                printf("\n The Dimensions of both matrices do not match.");
}
```

Chapter-3 Linked Lists

Very Short Answer Type Questions

Q.1

(i) What is linked list?
Ans. A linked list is a linear data structure in which each component (node) has at least two fields – data (information of component) and link (information about the location of next component. The end of the list is indicated by the special pointer constant NULL.

(ii) Give two applications of linked lists.
Ans. Two applications of linked lists are polynomial implementation and sparse matrix implementation.

(iii) Define traversing a linked list.
Ans. Traversing a linked list means processing each and every node of a linked list exactly once.

(iv) What do you mean by overflow and underflow in linked list?
Ans. When we want to insert a new element into a linked list and there is no memory space for the incoming element then it is said to be overflow in a linked list. And when we try to delete a element from an empty linked list then it is said to be underflow in a linked list.

(v) Define searching linked list.
Ans. By searching a linked list we mean finding an element in a linked traversing, starting from the first node of the linked list.

(vi) Define circular linked list.
Ans. A linear linked list in which each node has two fields – data (information of the node itself) and link (information about the location of next node) and the link field of last node contains the address of first node is called as a circular linked list.

(vii) Define doubly linked list.
Ans. A linked list in which each node contains two links, one for the next node and another for the previous one, is called as a doubly linked list.

(viii) Define Circular Doubly linked list.
Ans. A special type of doubly linked list in which one of the two link field of last node contain the address of previous node and another contains the address of first node. Similarly one of the two link field of first node contains the address of next node and another one contains the address of last node.

(ix) Write the difference between singly lined list and doubly linked list.
Ans. In a singly linked list, each node consists of an address of its next node; therefore traversing is done in one direction only. On the other hand, in a doubly linked list, each node consists of an address of its next node as well as its previous node, therefore traversing is done in both directions (forward as well as backward).

(x) Explain two advantages of array over linked list.
Ans. An array is easy to implement than a linked list. Traversing of array is faster than linked list.

Short Answer Type Questions

Q.2 Define a linked list. How linked list is represented in memory?.
Ans. A linked list is a linear data structure in which each component (node) has at least two fields – data (information of component) and link (information about the location of next component. The end of the list is indicated by the special pointer constant NULL.

A linked list can be represented in memory – either by using an array or by using pointers. Array is static in nature. Therefore pointers are frequently used in the implementation of a linked list.

Q.3 Write an algorithm to implement insertion into a linked list.
Ans. A linked list is a data structure in which the components are logically ordered by their pointer fields rather than physically ordered. There are two types of lists - Singly linked list (a list that traverses in one direction only) and doubly linked list (a list that traverses in both direction).

Insertion into a Linked List – Here is the algorithm that inserts an item in a linear linked list.

InsertList(First, Item, Pos)

Here 'First' contains either a NULL value or the address of first node of the linked list.

Step-1 : Create a new node 't'
Step-2 : Set the data field of 't' node as: data(t) = item;
Step-3 : if (First = NULL) or (pos==1) then Update the link field of 't' and 'First' as:
link(t) = Firs; and First = t;
And go to step-9; Otherwise Goto step-4
Step-4 : Initialize current and 'i' as:
current = First ;
i=1;
Step-5 : Repeat through step-7 while ((i<Pos-1) && (link(current) !=NULL))
Step-6 : Update the value of current as:
current = link(current)
Step-7 : Increment the value of 'i' as:
i = i+1

Step-8 : Update the link field of 't' and 'current' as:
link(t) = link(current)
link(current) = t;
Step-9 : Exit

Q.4 Write down the algorithm that deletes an element from the linked list.
Ans. **DeleteList(First, item)**

Here 'First' contains either a NULL value or the address of first node of the linked list. Perform the following steps if the list is not empty:

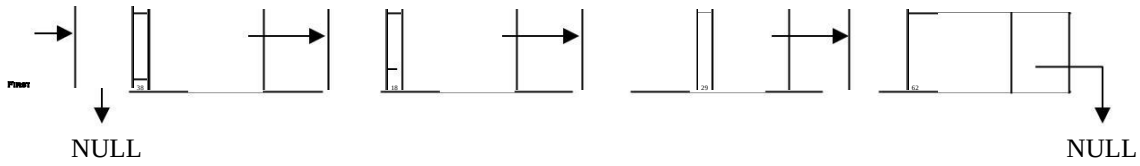
1. Set previous = NULL and current = First
2. Repeat through step-3 while ((current != NULL)&&(data(current) != item))
3. Update the value of previous and current pointers as:
previous = current
current = link(current)
4. Check when an item is found or not.
if (current == NULL) then display the message - "Item not found in the list."
and goto step-7
5. Check whether first node contains a deleted item:

if (previous == NULL) then update the value of First as First = link(First)
otherwise update the value of previous pointer as link(previous) = link(current)

6. Free the memory occupied by current as: Free(current)
7. Exit

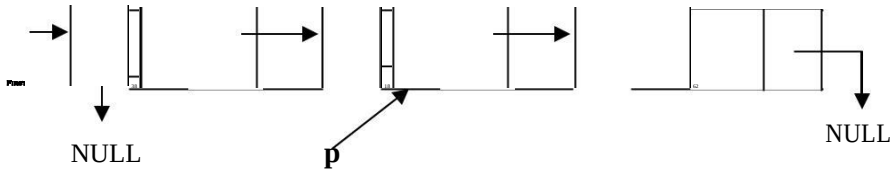
Q.5 What is Doubly linked list? How an element is inserted in a doubly linked list?

Ans. A linked list in which each node contains two links, one for the next node and another for the previous one, is called as a doubly linked list. Following figure shows a doubly linked list of 4 nodes.



Insertion of an element at any specific position - When we insert a new element (node) at any position in the doubly linked list we need to specify the position of the inserted node. If the position of new element is within the limitation then it is inserted at that specific position; otherwise this element is inserted at the end of the doubly linked list.

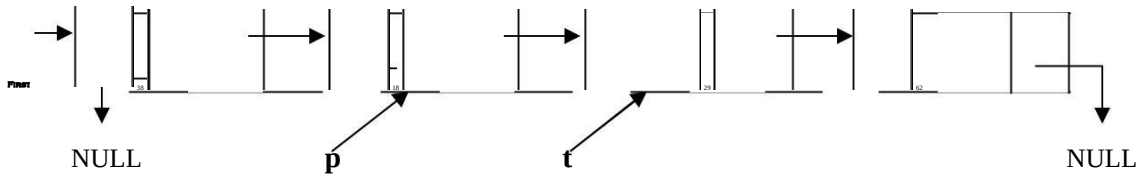
Let we have a linked list of three nodes as shown in following figure and we want to insert a new node 't' after the node 'p' .



When a new node 't' is inserted after 'p' node then the following link fields are changed accordingly:

data(t) = 29
rlink(t) = rlink(p)
llink(t) = p
llink(rlink(p) = t
rlink(p) = t

After using these statements, the doubly linked list might look like this:



Q.6 What is Garbage collection in linked list?

Ans. We know that a linked list stores a collection of elements non-contiguously. In a linked list, insertion or deletion of items is allowed anywhere. Also it is a dynamic data structure whose size can be arbitrarily modified. Therefore when such a dynamically created memory is no needed furthermore, the garbage collector reclaims this unused memory from such variable. In 'C' language this garbage collection is done automatically using the free() function.

Q.7 What is a node? How do you create a node of a linked list dynamically?
Ans. A node is an elementary item of linked list which contains the information of itself and the information of the location of its next node. A node of a linked list is created dynamically using malloc() function as:

```
struct node
{
        int data;
        struct node *link;
};
struct node *temp = (struct node *)malloc(sizeof(struct node));
```

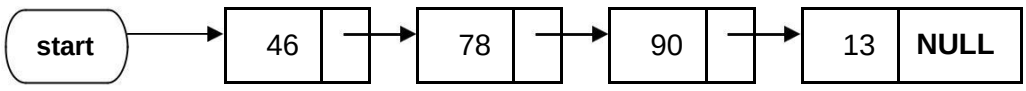
Q.8 How do you create a node of a doubly linked list dynamically?
Ans. A node in a doubly linked list contains of mainly three field - the information of itself and the information of the location of its next node and previous node. A node of a doubly linked list is created dynamically using malloc() function as:

```
struct dnode
{
        struct dnode *next;
        int data;
        struct dnode *next;

};
struct dnode *temp = (struct dnode *)malloc(sizeof(struct dnode));
```

Long Answer Type Questions

Q.9 What is a linked list? Explain representation of linked list in memory. Give its applications.
Ans. A linked list is a linear data structure in which each element contains two pieces of information - first part holds the *data* (information) of item and second part holds the address (*link*) of next element in the list. The entire list is accessed from an external pointer, start, that points (contains the address of) the first node in the list. The link field of last node in the list contains a special value, known as null, which is not a valid address; rather it just signifies the end of the list. Following figure shows this.



Representation of linked list in memory - A linked list can be represented in memory in two ways - using arrays and using dynamic variables (Pointers)

Array Representation of Linked Lists – As stated earlier, a linked list is basically a collection of elements and each element is also referred to as a node. Each array node consists an item of information, say ‘data’ item, and an integer variable, say link, containing the index of its successor node.

The declaration of an array node in a simple linked list is as follows:

```
struct ArrayNode
{
        int data;
        int link;
};
```

Here a pointer to a node is represented by an array index and a group of 10 nodes might be declared, globally, as:

```
#define n 100
struct ArrayNode node[n];
```

Here a pointer 'link' is an integer value between 0 and n-1, that references a particular element of the array node[]. In array implementation, node[i] refers to ith node, node[i].data is used to refer the data of node 'i' and node[i].link to next node of the linked list. The link field of last node contains a ***null*** pointer represented by the integer –1.

However, it is not necessary that the linked list from node[0], rather it can be started from any node number, say node[3]. Since a variable 'start' represents a pointer to the list, therefore 'start' points to node[3] as it is the first node on the list. The node[3].data references the information of the current node and node[3].link contains the index value of second node. Let node[3].link is 8 then node[8] is the second node of the list. The node[8].link will contain the index of value of third node. This process goes on until the linked list is completed. The index value of last node of the list is represented by the integer –1. The value –1 is just used to represent the end of the list. However we can also used any unusual values, such as -9999 or -11111 etc, for representing the end of the list.

Pointers Representation of Linked Lists Using Dynamic Variables (Pointers)

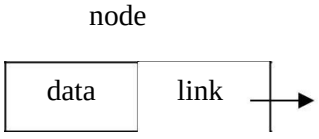
As stated earlier each node of a linked list has two fields – the data (information) field and a link field pointing to the next node in the list. Nodes are created dynamically using dynamic memory allocation function ***malloc()***, when needed, and destroyed using ***free()*** function, when no longer needed. The link field contains the memory address of the next node in the list. Such a node can be declared in C using self-referential structure.

Here the link field of each node, except last node, contains the memory address of its successor node. The link field of the last node contains sentinel value, defined as NULL in C.

A typical node of a linked list is declared as:

```
struct node
{
        int data;
        struct node *link;
};
typedef struct node node;
```

and it is represented as:



A node of this type is identical to the nodes of the array implementation except that the link field is a pointer (containing the address of the next node in the list) rather than an integer containing the index within an array where the next node in the list is kept.

To maintain a linear linked list we use a pointer variable '*first'*, also known as an *external pointer* variable. The 'first' pointer contains an address of first node in the list. Here the name 'first' is just an user-defined identifier. You can also use other names, such as start, or head etc. if the list is empty then the value of this external pointer variable must contain a NULL value.

When a linked list is implemented using pointers, nodes are allocated and free as necessary. Thus there is no need to declare a collection of nodes. A new node is created dynamically as:

```
node *p = (node *)malloc(sizeof(node));
```

This statement places the address of an available node into 'p'. Here the malloc() function is used to create memory dynamically. Similarly the memory allocated to 'p' node is released by using free() function as:

```
free(p);
```

Q.5 Write a program which a singly (linear) linked list of 'n' number of nodes and traverses it.

```
#include <stdio.h>
#include <alloc.h>
struct node
{
        int data;
        struct node *link;
};
typedef struct node node;
main()
{
        int i,n, item;
        node *t, *current;
        node *first = NULL;

        printf("\nHow many nodes you want to create? - ");
        scanf("%d", &n);
        for(i=1; i <=n; i++)
        {
                printf("Enter the data field of node-%d = ",i);
                scanf("%d", &item);

                t = (node *)malloc(sizeof(node));
                t->data = item;
                t->link = NULL;

                if (first == NULL)
                        first = t;
                else
                        current->link = t;
                current = t;
        }
        printf("\nLinked list is as....\n\n");
        while (first != NULL)
```

```
        {
                printf("%d —> ", first->data);
                first = first->link;
        }
        printf("Null");
}
```

Q.6 Write an algorithm for searching an element in a sorted liked list.

Ans. Here is the algorithm for searching an element in sorted linked list:

Search(First, Item)
Here 'First' contains the address of first node and Item is the element to be searched in it.

Step-1 : Set Flag = 0
Step-2 : Repeat through step-4 while (First != NULL)
Step-3 : Compare Data(First) and Item
        If Data(First) = Item then
                Set flag = 1 and
                go to step-5
Step-4 : Update the value of First as First = Link(First)
Step-5 : If Flag = 1 then print – "Item is found"
        Otherwise print – "Item not found"
Step-6 : Exit

Q.7 Write a program which a doubly (linear) linked list of 'n' number of nodes and traverses it.

Ans. Here is a program which a doubly (linear) linked list of 'n' number of nodes and traverses it.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
struct dnode
{
        struct dnode *left;
        int data;
        struct dnode *right;
};
typedef struct dnode dnode;
main()
{

        dnode *first = NULL;
        dnode *temp, *current;
        int i, item, n;

        printf("\nCreating a Doubly Linked List.\n");
        printf("\nHow many nodes you want to create? :");
        scanf("%d", &n);
        printf("\n");
        for(i=1; i<=n; i++)
        {
                printf("Enter the data field of node #%d :", i);
                scanf("%d", &item);
                temp = (dnode *)malloc(sizeof(dnode));
```

```
                temp->data = item;
                temp->right = NULL;

                if (first == NULL)
                {
                        first = temp;
                        temp->left = NULL;
                }
                else
                {
                        current->right = temp;
                        temp->left = current;
                }
                current = temp;
        }

        if (first = = NULL)
                printf("\nList is empty.\n");
        else
        {
                printf("\nTraversing in forward direction.\n\n");
                while (first->right != NULL)
                {
                        printf("%d<—>", first ->data);
                        first = first ->right;
                }
                printf("%d <—> Null", first ->data);
                printf("\n\nTraversing in backward direction.\n\n");
                while (first->left != NULL) {

                        printf("%d<—>", first ->data);
                        first = first ->left;
                }
                printf("%d <—> Null", first ->data);
        }
}
```

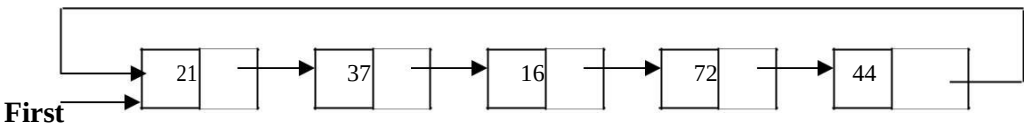
Q.8 Write short notes on

(a) Circular linked list

(b) Header linked list

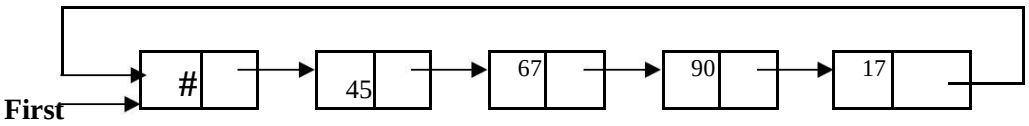
Ans.

(a) Circular linked list – Circular linked list is a special type of singly linked list in which each node has two fields – data and link. The data field contains the information of the node and the link field contains the information of the next node in the list. The special thing about it is that the link field of last node contains the address of first node rather than a NULL value like in a linear linked list. Following figure shows this:



Circular linked queues have two basic operations – Insertion and Deletion and four other important operations, such as InitializeQueue, FrontElement, IsQueueFull and IsQueueEmpty. Here we will only discuss insertion and deletion operations. Rest are left for the readers.

(b) Header linked list – A header linked list is also a special type of linked list in which there is always a special designated first node, called a *header* node, at the front of the list. This header node contains some special and important information of list. For example, a header node usually contains information such as the total number of nodes in the list or whether the list is sorted or not.



Generally we have two types of header linked lists:

(i) **Grounded Header Linked Lists -** A grounded header linked list is one whose last node contains a NULL value

(ii) **Circular Header Linked Lists -** A circular leader linked list is one whose last node points to the header node; rather than containing a NULL value.

Chapter-4 Stacks, Queues and Recursion

Very Short Answer Type Questions

Q.1

(i) What is stack? Give example.
Ans. A stack is a linear data structure in which insertions and deletions are made at one end, called top of the stack. A stack is called as LIFO (Last–In First –Out) structure because the item that is inserted in last will be the first item to be removed from the stack.

(ii) How do you represent stack in computer?
Ans. A stack is represented in computer by using either an array or a linked list (pointers).

(iii) What is Push and Pop in stack?
Ans. When a new item is inserted into the stack it is said to be PUSH operation. And when an item is deleted from the stack it is said to be POP operation.

(iv) Give two uses of stack.
Ans. Two applications of stacks are:
(a) Stacks are more commonly used within the computer system when functions are called.
(b) Stacks are also used in evaluating of postfix expressions.

(v) Complete the term LIFO and FIFO.
Ans. LIFO stands for Last In First Out and
FIFO stands for First In First Out

(vi) Define linear queue.
Ans. A linear queue is a data structure in which all insertion to the list are made at one end, called the ***rear***, and all deletions from the list are made at the other end, called the ***front***. In a linear queue, both *rear* and *front* are incremented upon insertion and deletion operations respectively.

(vii) What is polish notation?
Ans. The process of writing the operators of an expression either before their operands or after them is called '***Polish'*** notations.

(viii) What is prefix notation of A – B / C ?
Ans. The prefix notation of A – B / C is
– A / B C

(ix) What do you mean by Postfix notation or Reverse polish notation?
Ans. A notation in which operators comes after its operands is called as postfix notation. For example - a b + is postfix notation of a + b .

(x) Define a deque or De-queues.
Ans. The deque (**d**ouble **e**nded **que**ue) is an ordered collection of elements from which new elements can be added or deleted from either end of the queue but not in the middle.

(xi) Define priority queue.
Ans. A queue in which items are inserted according to their priority and deleted from the front of the queue is known as a priority queue.

(xii) Front and rear are used with outline (T/F).
Ans. True

(xiii) What do you mean by recursion?
Ans. A recursion is a process in which a function calls itself directly or indirectly.

Section-B

Short Answer Type Questions

Q.2 Explain Push operation on stack. Also write algorithm to this operation.
Ans. When we insert an item onto a stack, we say that we push it onto the stack. A new item is inserted into a stack only when the stack is not full; otherwise it displays an error message – "Stack Overflow". Therefore when a new item is inserted, we will firstly check, whether the stack is full or not. If the value of top is equal to (MAXSTK-1) then our stack is overflow; otherwise the top is incremented by one and the item is inserted at current top position of the stack.

Algorithm - PUSH(STACK, ITEM)
Here STACK is an array of items and ITEM is the new item to be inserted

1. Check whether the STACK is full of not
   If (TOP = = MAXSTK - 1) then
      Display – "Stack Overflow." And go to Step-4
2. Increment TOP as TOP = TOP + 1
3. Place the new ITEM at TOP position as:
   STACK[TOP] = ITEM
4. Exit

Q.3 In the given stack perform the operation PUSH(STACK, WWW)

XX	**YY**	**ZZ**					
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**

Top = 3 **MAXSTK = 8**

Ans. Push(Stack, WWW) – In this first the value of TOP is increased and then the item 'WWW' is pushed as:

Top = Top + 1
STACK[Top] = WWW

Now the STACK might looks like this:

XX	YY	ZZ	WWW				
1	2	3	4	5	6	7	8

Top = 4 MAXSTK = 8

Q.4 What are infix, postfix and prefix notations for an arithmetic expression? Explain with suitable examples.
Ans. A notation in which operators comes between its operands in an arithmetic expression is called infix notation. When an operator comes after its operands, it is called as postfix notation. And when an operator comes before its operands, it is called as prefix notation Here are examples of infix, postfix and prefix notation for an arithmetic expression (infix) a + b:

a b + is postfix notation of a + b .
+ a b is prefix notation of a + b

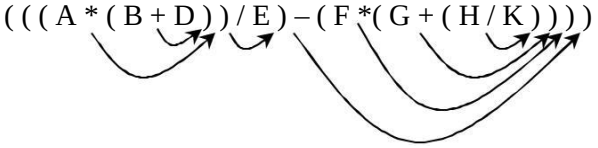
Q.3 Write the postfix and prefix forms of:

A * (B + D) / E – F *(G + H / K)

Ans. Firstly we will parenthesize this expression according to the operator priority precedence as:

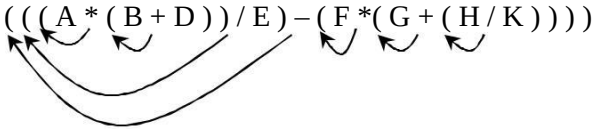
(((A * (B + D)) / E) – (F *(G + (H / K))))

Now for the post fix expression shift all the operators to their respective ending brackets and remove all the parenthesis as:



Postfix expression : A B D + * E / F G H K / + * –

For prefix expression shift all the operators to their respective opening brackets and remove all the parenthesis as:



Prefix expression : – / * A + B D E * F + G / H K

Q.4 How the traversing of stack performed?
Ans. While traversing a stack each element of a stack is processed exactly once from top position to bottom one. Here is the algorithm that traverses a stack.

Traverse(Stack)
Here 'Stack' contains the base address of the array 'Stack'.
Perform the following steps if the stack is not empty:

1. Set I = Top
2. Repeat through step-4 while (I>=0)
3. Process the element Stack[I]
4. Decrement I as I = I – 1
5. Exit

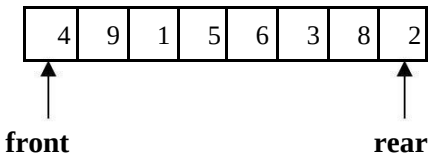
Q.5 Explain the following notations:
(i) infix
(ii) Prefix
(iii) Postfix

Ans. Infix notation – If an operator is placed between its two operands in an expression, it is said to be an '*infix'* expression and this notation is called as *infix* notation. For example: a+b, a*b etc.

Prefix notation – If an operator is placed just before its operands, it is said to be a prefix notation. For example +a b, *a b etc.

Postfix notation – If the operator is placed after its two operands, it is said to be a *postfix* notation. For example a b +, a b * etc.

Q.6 Explain Queues. How you will present and implement the queue in memory.
Ans. A queue is linear data structure in which all insertion to the list are made at one end, called the ***rear***, and all deletions from the list are made at the other end, called the ***front***. Following figure shows a queue of 8 elements.



The queue is also known as FIFO (First In First Out) data structure because the first element that is inserted into the queue would be the first one to be removed.

In C, a queue is implemented either by using an array or by using a linked list (pointers).

Q.7 Explain Insert operations in a queue.
Ans. Let Queue is an array of 'N elements and front and rear contain the index value of front and rear elements respectively. Initially front and rear both are set to -1.

Insert Operation – An item is inserted into the queue only when the queue is not full. Insert operation is implemented as:

Insert(Queue, item)
Here 'Queue' contains the base address of the array and 'item' be the element to be inserted.

Step-1 : If front = N – 1 then display an error message - "Queue Overflow" and go to step-5; otherwise go to step-2
Step-2 : Increment the value of 'rear' as rear = rear+1
Step-3 : Store the 'item' at rear position as Queue[rear]=item
Step-4 : Set front = 0 only when (front == -1)
Step-5 : Exit

Q.8 Explain Delete operations in a queue.
Ans. An item is deleted from the queue only when the queue is not empty. Delete operation is implemented as:

Delete (Queue)
Here 'Queue' contains the base address of the array Queue.

Step-1 : If front = -1 then display an error message - "Queue Underflow" and go to step-4; otherwise go to step-2
Step-2 : Access the front element as item = Queue[front]
Step-3 : Reset the value of front and rear if (front == rear);
Otherwise increment the value of front as
front = front+1
Step-4 : Exit

Q.9 What are circular queues? Explain.

Ans. A circular queue is a special queue in which front and rear counter variables are moved in a circle. When a circular queue is implemented it thinks of an array as a circle rather than a straight line. In this type of queue, the rear moves from last position to first position, only when there is a vacant place at the beginning of the array. Similarly the value of front moves from (Size-1) to 0, if it is possible. In circular queue, the value of rear is incremented as:

```
rear = (rear+1)%Size;
```

Similarly the value of front is incremented as:

```
front = (front+1)%Size;
```

Q.9 Write down various operations performed on circular queues.

Ans. InitializeCQueue(CQ) – Initializes a circular queue CQ as an empty queue.
InsertQueue(CQ, Item) – Inserts an 'Item' into a circular queue
DeleteQueue(CQ) – Deletes front element of the queue
IsCQueueFull(CQ) – checks whether the circular queue is full or not
IsCQueueEmpty(CQ) – checks whether the circular queue is empty or not

Q.10 Write a program that finds the factorial of any given number using recursion.

Ans. Here is a program that finds the factorial of any given number using recursion.

```
#include <stdio.h>
main()
{
        int num , n ;
        printf("\nEnter the number : ") ;
        scanf("%d",&n);
        num = factorial(n);
        printf("\nThe factorial of a given number %d =  %d", num , number ) ;
}
factorial(int n)
{
        if ( n = = 0 )
                return (1) ;
        else
                return (n * factorial (n-1)) ;
}
```

Q. 11 Write a program in C to calculate the Fibonacci series using recursion.

Ans. Here is a program in C to calculate the Fibonacci series using recursion.

```
#include <stdio.h>
main ()
{
        int n;
        printf ("\n Enter the number of terms to be generated?");
        scanf ("%d", &n);
        printf ("\n Fibonacci series upto %d terms \n",
        number); fib (n);
}
fib (int n)
{
```

```
        if ((n= =0) || (n= =1))
                return 0;
        else
        {
                if (n= =2)
                        return (1);
                else
                {
                        f = fib (n-1)+fib (n-2);
                        printf ("%d",f);
                }
        }
}
```

Q.12 What do you mean by iteration? Compare recursion with iteration.

Ans. Iteration is a process of repeating a set of statements over and over till a condition is true. Iterations are called as loops in high level programming languages. C provides three types of iterative statements:

1. for loop
2. while loop
3. do-while loop

On the other hand, recursion is a process in which a function calls itself directly or indirectly. Recursion is a fast process as compare to iterations.

Recursion is preferred over iteration when the problem is naturally recursive. For example, finding a factorial of a given number is naturally recursive. On the other hand, iteration is not always overshadow by recursion. Recursion becomes very dangerous if it is used improperly.

Long Answer Type Questions

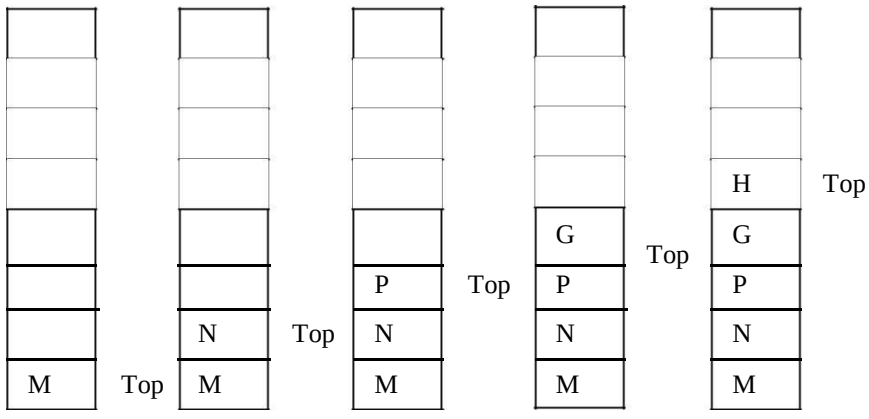
Q.12 What is stack? Explain with example.

Or

Explain Push and Pop in stack with a suitable example.

Ans. Stack is a data structure in which all insertion and deletion are made at only one end, called the top of the stack. When a new item is inserted into the stack it is said to be PUSH operation. And when an item is deleted from the stack it is said to be POP operation.

Let we have a stack 'S' of 8 cells, which is initially empty and we want to insert the following sequence of characters - 'M', 'N', 'P', 'G', 'H'. Here 'M' is the first character to put on the stack and 'H' is the last character as shown in following figure.



If we want to delete an item from stack then the first element to be deleted would be 'H'. You can also say that the element, which is most recently put on the stack, will be the first one to be removed. That's why Stack is also known as LIFO, which stands for Last In First Out.

There are two major of operations performed on stack:

Push Operation - When we insert an item onto a stack, we say that we push it onto the stack. A new item is inserted into a stack only when the stack is not full; otherwise it displays an error message – "Stack Overflow". Therefore when a new item is inserted, we will firstly check, whether the stack is full or not. If the value of top is equal to (MAXSTK-1) then our stack is overflow; otherwise the top is incremented by one and the item is inserted at current top position of the stack.

Algorithm - PUSH(STACK, ITEM)
Here STACK is an array of items and ITEM is the new item to be inserted

1. Check whether the STACK is full of not
   If (TOP = = MAXSTK - 1) then
      Display – "Stack Overflow." And go to Step-4
2. Increment TOP as TOP = TOP + 1
3. Place the new ITEM at TOP position as:
   STACK[TOP] = ITEM
4. Exit

Pop Operation - When we delete an item from stack, it is said to be pop operation. An item is deleted from the stack only when the stack is not empty. If the stack is empty then it displays an error message – "Stack Underflow"; otherwise the top element of the stack is removed and the value of TOP is decremented.

Algorithm - POP(STACK)
Here STACK is an array of items

1. Check whether the STACK is empty of
   not If (TOP = = -1) then
      Display – "Stack Underflow." And go to Step-4
2. Remove t he top element and store it in ITEM as:

3. Decrement TOP as TOP = TOP - 1
4. Exit

Q.14 What do you understand by Precedence of arithmetic operators. Convert the following expression into Postfix form using Stack:
A / B + C / D * (E + F) / H

Ans. When we convert an infix expression into postfix expression, we scan each symbol from left to right. If an operand is encountered then it is appended to postfix expression. If a left parenthesis is encountered then it is pushed into stack and when right parenthesis is encountered then all operators upto the first left parenthesis are popped from the stack. However if an operator is scanned then it follows the rule:

(i) If the scanned operator has lower or equal priority as compared to the top of the operators on stack then the operator on top of the stack is popped from the stack and is appended to the postfix expression and then the new scanned character is pushed on to the stack

(ii) If the scanned operator has higher priority as compared to the top of the operator on stack then the scanned operator is pushed on top of the stack.

Scanned Symbol	**Stack**	**Postfix**
(	(	
A	(	A
/	(/	A
B	(/	A B
+	(+	A B /
C	(+	A B / C
/	(+ /	A B / C
D	(+ /	A B / C D
*	(+ *	A B / C D /
(	(+ * (	A B / C D /
E	(+ * (	A B / C D / E
+	(+ * (+	A B / C D / E
F	(+ * (+	A B / C D / E F
)	(+ *	A B / C D / E F +
/	(+ /	A B / C D / E F + *
H	(+ /	A B / C D / E F + * H
None	Empty	A B / C D / E F + * H / +

Thus the postfix expression of the given infix expression is A B / C D / E F + * H / +

Q. 15 Consider the following stack where Stack is allocated N = 4 memory cells:

STACK: AAA, BBB, ___, ___

Describe the stack as the following operations take place:

(a) POP(STACK, ITEM)
(b) POP(STACK, ITEM)
(c) PUSH(STACK, HHH)
(d) POP(STACK, ITEM)
(e) POP(STACK, ITEM)
(f) PUSH(STACK, GGG)

Ans. (a) POP(STACK, ITEM) – removes the topmost item BBB from the stack and store it in ITEM and the STACK looks

STACK: AAA, ___, ___, ___

(b) POP(STACK, ITEM) – removes the topmost item AAA from the stack and store it in ITEM and the STACK looks

STACK: ___, ___, ___, ___

(c) PUSH(STACK, HHH) – inserts new item HHH on top of the stack and the STACK looks

STACK: HHH, ___, ___, ___

(d) POP(STACK, ITEM) – removes the topmost item HHH from the stack and store it in ITEM and the STACK looks

STACK: ___, ___, ___, ___

(e) POP(STACK, ITEM) – displays an error message – "STACK UNDERFLOW" because there is no item in the stack to be removed

(f) PUSH(STACK, GGG) – inserts new item GGG on top of the stack and the STACK looks

STACK: GGG, ___, ___, ___

Q.17 Consider the following queue of characters where QUEUE is a circular array which is allocated 6 memory cells:

FRONT = 2, REAR = 4

QUEUE = ___, A, C, D, ___, ___

Describe the queue for the following operations:

(a) F is added to the queue
(b) Two letters are deleted
(c) K, L and M are added to the queue
(d) Two letters are deleted
(e) R is added to the queue

Ans. We have a circular queue of 6 memory cells as:

Front = 2, Rear = 4, and

QUEUE= _, A, C, D,__,__

(i) When F is added to the queue, the rear changes to 5 and the Queue is

QUEUE= _, A, C, D, F,__ (Front = 2, Rear = 5)

(ii) When two letters are deleted, the front changes to 4 and the Queue is

QUEUE= _, _, _, D, F, _ (Front = 4, Rear = 5)

(iii) When K, L and M are added to the queue, the rear changes to 2 and the Queue is

QUEUE= L, M, _, D, F, K (Front = 4, Rear = 2)

(iv) When two letters are deleted, the front changes to 6 and the Queue is

QUEUE= L, M, _, _, _, K (Front = 6, Rear = 2)

(v) When R is added to the queue, the rear changes to 3 and the Queue is

QUEUE= L, M, R, _, _, K (Front = 6, Rear = 3)

Q.18 Consider the following circular queue capable of accommodating maximum Six elements

FRONT = 2, REAR = 4

QUEUE = ___, A, B, C, ___, ___

Describe the queue for the following operations:

(a) Add N
(b) Add Z
(c) Delete two letters

(d) Add M, H, I
(e) Delete one letter

Ans. We have a circular queue of 6 memory cells as:

Front = 2, Rear = 4, and

QUEUE=__,A, B, C,__,__.

(i) When N is added to the queue, the rear changes to 5 and the Queue is

QUEUE= __, A,B, C, N,__. (Front = 2, Rear = 5)

(ii) When Z is added to the queue, the rear changes to 6 and the Queue

is QUEUE= __, A,B, C, N, Z (Front = 2, Rear = 6)

(iii) When two letters are deleted, the front changes to 4 and the Queue is

QUEUE = __, __, __, C, N, Z (Front = 4, Rear = 6)

(iv) When M, H and I are added, the rear again changes to 3 and the Queue is

QUEUE = M, H, I , C, N, Z (Front = 4, Rear = 3)

(e) When one letter is deleted , the front changes to 5 and the Queue is

QUEUE = M, H, I , __, N, Z (Front = 5, Rear = 3)

Q.19 Write the short notes on

a. Stack
b. Priority Queue
c. De-Queues
d. Recursion
e. Two applications of Stack

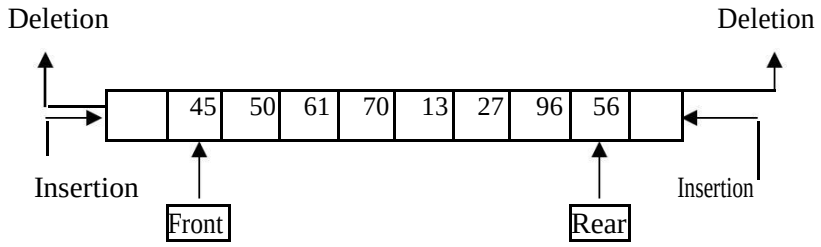
Ans. (a) Stack - A stack is a basic data structure, where insertion and deletion of items takes place at one end called top of the stack. The basic concept can be illustrated by thinking of your data as a stack of plates or books where you can only take the top item off the stack in order to remove things from it.

A stack is also called a LIFO (Last In First Out) to demonstrate the way it accesses data. There are basically three operations that can be performed on stacks . They are 1) inserting an item into a stack (push). 2) deleting an item from the stack (pop). 3) displaying the contents of the stack(pip).

(ii) Priority Queue - A priority queue is a data structure in which elements are inserted arbitrarily but deleted according to their priority. If elements have an equal priority, then the usual rule applies, that is first element inserted should be removed first. Priority queues are of two types:

(a) Ascending Priority Queues - In *ascending priority queues*, the elements are inserted arbitrarily at any position depending upon their priority but delete the element having smallest priority.
(b) Descending Priority Queues - In *descending priority queues*, the elements are inserted arbitrarily at any position depending upon their priority but delete the element having largest priority.

(iii) De-Queues – The Deque stands for Double Ended Queue. A deque is an ordered collection of elements from which new elements can be added or deleted from either end of the queue but not in the middle, as shown in following figure



In this figure, there are 8 elements in the deque. Here the new element can be inserted at either end of the queue. Similarly the elements can be removed from the either end of the queue. Deques are of two special types -

1. **Input Restricted Deque** – a deque in which items may be deleted at either end, i.e. front as well as rear, but insertion of items is restricted at only one end, say rear, of the queue.

2. **Output Restricted Deque** – a deque in which items may be inserted at either end, i.e. front as well as rear, but deletion of items is restricted at only one end, say front, of the queue.

(iv) Recursion – Recursion is a process in which a functions calls itself directly or indirectly. While using recursion one should remember that there should be at least one if statement used to terminate recursion. It does not contain any looping statements. For example:

```
Recursion()
{
        printf("\n Recursion….!");
        Recursion();
}
```

Advantages : (i) It is easy to use (ii) It represents compact programming structures.
Disadvantages : It is slower than that of looping statements because each time function is called.

(v) Two application of stacks -

(a) Stacks are more commonly used within the computer system when functions are called. A function may be called either by itself or by other function. When a function is called within another function then the system should remember the location from where the call was made, additionally the parameters and local variables of the calling function so that their values will not be lost while its execution resumes.
(b) Stacks are also used in evaluating of postfix expressions.

Very Short Answer Type Questions

Q.1

(i) Define a tree.

Ans. A tree, 'T', may be defined as a finite set of one or more nodes such that

a) there is a specially designated node, called the root, 'R', of the tree
b) the remaining nodes (excluding the root node) are partitioned into $n \geq 0$ disjoint sets T1, T2, …., Tn and each of these sets is a tree in turn. The trees T1, T2, …., Tn are called the *subtrees* of the root.

(ii) Define binary tree.

Ans. A *binary tree* 'T' is defined as a finite set of nodes that is either empty or consists of one or more nodes such that there is a specially designated node, called the *root* of the tree, R, and two disjoint binary trees called the left subtree and the right subtree. A left or right subtree can be empty.

(iii) What is the significance of the root node?

Ans. The root node is a specially designated node from where the traversing of a tree starts.

(iv) Define how do you represent a binary tree in memory.

Ans. A binary tree can be represented in two ways - sequential representation (array) and linked list representation.

(v) Define degree and non-terminal node.

Ans. The number of subtrees of a node is called its *degree*. The node that has one or more subtrees are called as non-terminal node.

(vi) Define leaf nodes (terminal nodes) and edges.

Ans. The nodes that have degree zero are called *leaf* nodes. Leaf nodes are also called as terminal nodes. The links between a parent and its children are also referred to as *edges* or *branches*. If 'X' is a father of 'Y' then we can define an edge as a tuple (X, Y).

(vii) Define edge.

Ans. The link between a parent node and its child node is called an edge.

(viii) Define path.

Ans. A collection of edges connected one node to another is often termed as a path.

(ix) Define height and path of a binary tree.

Ans. The height of a binary tree is defined as the length of the longest path starting at the root and a path is a collection of edges connected one node to another.

(x) Name various operations for binary tree.

Ans. Traversing, insertion, deletion and searching are major operations for a binary tree.

(xi) Write the steps used in preorder traversing of a binary tree.

Ans. In Preorder, the root is visited before (pre) the subtrees traversals. Its algorithm is as:

1. Visit the root
2. Traverse left subtree
3. Traverse right subtree

(xii) What is a heap?

Ans. A heap is defined to a complete binary tree 'H' such that each node of H has the following property – the value at any node, say 'i' is greater than or equal to the value of at any of its descendants. Such a heap is called as *maxheap*. On the other hand if the value at any

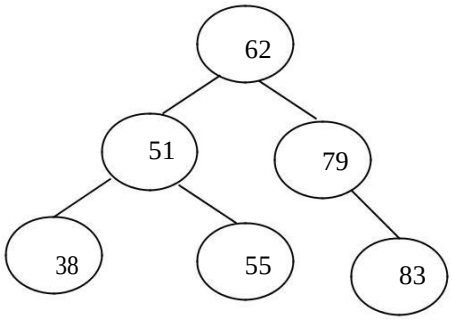
node, say ‘i’ is less than or equal to the value of at any of its descendants then it is called as *minheap*.

Short Answer Type Questions

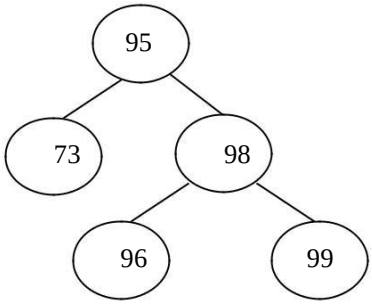
Q.2 What do you mean by binary search tree (BST) ? Explain it with example.
Ans. A binary search tree (BST) is a special binary tree and is defined as:

(i) every node has a key and no two nodes have the same key value
(ii) all left subtree’s keys (if any) are smaller than the root’s key
(iii) all right subtree’s keys (if any) are larger than the root’s key
(iv) the left and right subtrees of a binary search tree are itself binary search trees

Following figure shows some example of binary search trees.



(a)

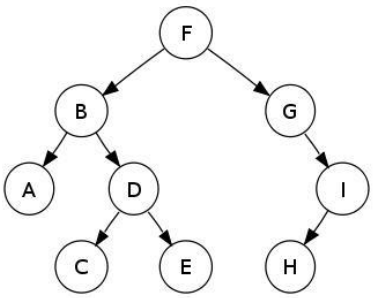


(b)

Q.3 Explain Pre order traversing technique using suitable example.
Ans. To traverse a non-empty binary tree in **pretorder**, perform the following operations recursively at each node:

1. Visit the root.
2. Traverse the left subtree.
3. Traverse the right subtree.

Let we have a following binary tree:



The preorder traversal of this binary tree is: F, B, A, D, C, E, G, I, H

Q.3 Write algorithm of Search operation in binary search tree.
Ans.

Algorithm: Search(Root, item)
Here Root contains the address of root node and item be the element to be searched.

Step-1 : If (Root == NULL) then return 0;

Step-2 : If (Root->data == item) return 1
Else if (Root->data > item) Call Search(Root->lchild, item)
Else Call Search(Root->rchild, item)
Step-3 : Exit

Long Answer Type Questions

Q.4 How is binary tree represented in memory? Explain.
Or
Explain sequential representation and linked list representation of a binary tree using suitable example.

Ans. A binary tree is a specific type of tree in which each node can have a maximum of two children. These child nodes are typically distinguished as the left and the right child. The tree made up of a left child (of a node x) and all its descendents is called the left subtree of x. Similarly we can define right subtrees.

The structure of a binary tree is shown in figure 3.

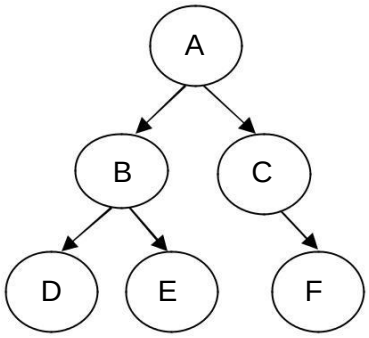


Figure-3

In the above binary tree, node B is the left child of node 'A', and node C is the right child of node 'A'. Similarly D and E are the left and right child's of node B.

Representing a Binary Tree

Binary Trees can be implemented by using arrays as well as linked lists.

Array (Sequential) Representation

In this method, the nodes of a binary tree are represented as elements in an array. The node started in array are accessible sequentially. The root node always lies at index '0'. For example:
The array representation of a binary tree is represented in an array by storing each element at the array position corresponding to the number assigned to it.

0 1 2 3 4 5 6

The binary tree to be represented is regarded as a complete binary tree with some missing elements.
example:

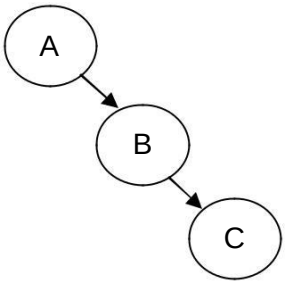
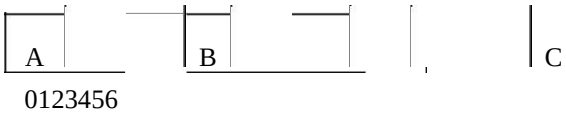
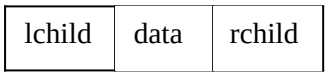


Figure-4

The array representation of figure-4 is as:



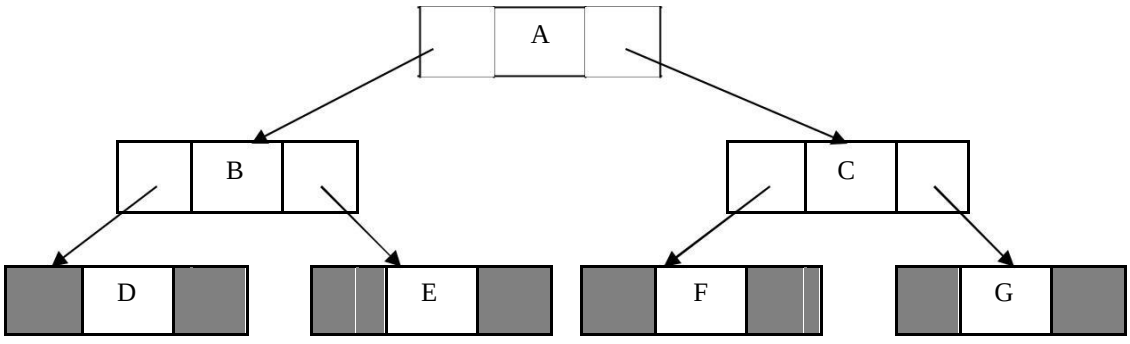
Structure of Tree Node



Syntax:

```
struct tnode
{
        struct tnode *lchild;
        int info;
        struct tnode *rchild;
};
```

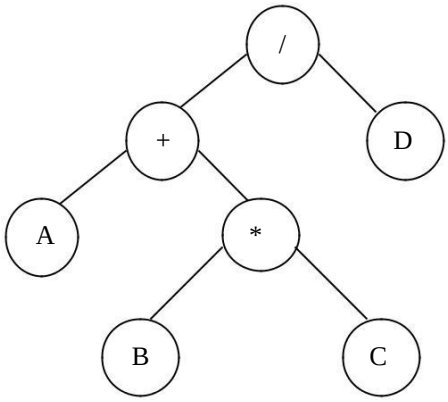
The linked representation of figure-3 is:



Q.5 What do you mean by traversing a binary tree? Explain various types of traversing techniques and write algorithm for preorder traversal and implement the same using 'C'.

Ans. Traversal is the process of visiting every node in a tree data structure, exactly once, in a systematic way. Such traversals are classified by the order in which the nodes are visited. If there are 'n' nodes in a binary tree then there are n! different orders in which they could be visited. However a good traversal technique visits the nodes of a tree in some linear sequence. When a node is picked then there are three tasks to do at a given node - visit the node itself (V), traverse its left subtree (L) or traverse its right subtree (R).

Generally the left subtree always precedes visiting the right subtree, thus there are three possible traversal techniques known as inorder (LVR), postorder (LRV) and preorder (VLR) respectively. Consider a simple expression tree.



As discussed earlier, there are three ways in which you can traverse a binary tree. They are:

a) **Preorder** - In Preorder, the root is visited before (pre) the subtrees traversals.

Algorithm: Preorder(Root)
Here 'Root' contains the address of the root node. Perform these steps if Root is not equal to Null.

1. Print the value of data of Root node
2. Call Preorder(lchild(Root))
3. Call Preorder(rchild(Root))
4. Exit

'C' Implementation:

```
Preorder(TreeNode *Root)
{
        if (Root != NULL)
        {
                printf("\n%c", Root->data);
                Preorder(Root->lchild);
                Preorder(Root->rchild);
        }
}
```

If this function is executed on the tree of shown in above figure then the following output would be printed:

/ + A * B C D

b) **Inorder -** In Inorder, the root is visited in-between left and right subtree traversal.

Algorithm: Inorder(Root)
Here 'Root' contains the address of the root node. Perform these steps if Root is not equal to Null.

1. Call Inorder(lchild(Root))
2. Print the value of data of Root node
3. Call Inorder(rchild(Root))
4. Exit

'C' Implementation:

```
Inorder(TreeNode *Root)
{
        if (Root != NULL)
        {
                Inorder(Root->lchild);
                printf("\n%c", Root->data);
                Inorder(Root->rchild);
        }
}
```

If this algorithm is used then the following output would be printed:

A + B * C / D

c) **Postorder** - In Postorder, the root is visited after (pre) the subtrees traversals.

Algorithm: Postorder(Root)
Here 'Root' contains the address of the root node. Perform these steps if Root is not equal to Null.

1. Call Postorder(lchild(Root))
2. Call Postorder (rchild(Root))
3. Print the value of data of Root node

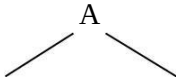
'C' Implementation:

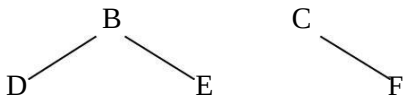
```
Inorder(TreeNode *Root)
{
        if (Root != NULL)
        {
                Inorder(Root->lchild);
                printf("\n%c", Root->data);
                Inorder(Root->rchild);
        }
}
```

If this function is executed on the tree shown earlier then the following output would be printed:

A B C * D / +

Q. 6 Consider the given tree and write its preorder, inorder and postorder





Ans.

Inorder Traversal – D B E A C F
Preorder Traversal – A B D E C F
Postorder Traversal – D E B F C A

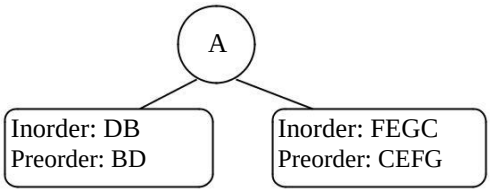
Q. **7 Construct a binary tree with the following inorder and preorder traversal.**

Inorder Traversal : D B A F E G C

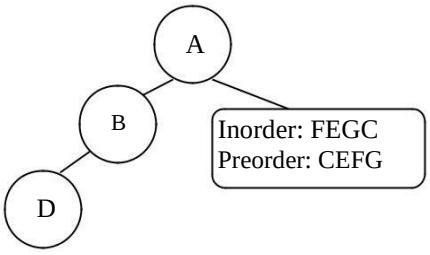
Preorder Traversal : A B D C E F G

Ans. We know that in preorder traversal of a binary tree, the root node is visited first, therefore the preorder sequence is looked for the node to be visited. And as far as the inorder traversal is concerned, the root node is visited after traversing the left sub-tree. Therefore the inorder sequence is looked for those node who are in the left subtree of the visited node and who are in the right subtree of the visited node.

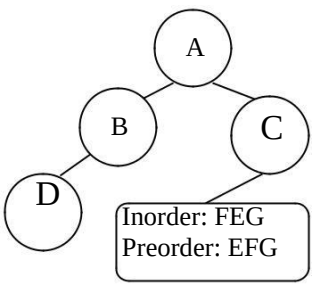
And since 'A' is the first node in the preorder sequence, therefore 'A' becomes the root node of the binary tree. And as far as the inorder traversal is concerned, the root node is visited after traversing the left sub-tree. Therefore all the nodes which are on the left side of 'A' in the given inorder sequence belong to the left subtree and the nodes to the right of 'A' belong to the right subtree of the tree as shown below:

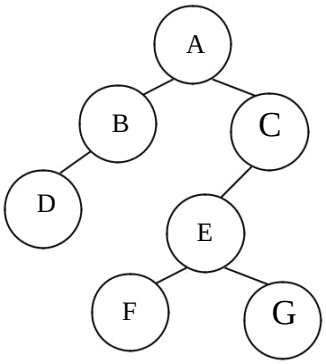


Again if there are 'n' number of nodes after 'A' in inorder sequence then the first 'n' nodes after 'A' in preorder sequence become the preorder sequence of the left subtree. This process is applied to both the left and right subtrees once again. Let we consider the left subtree. The preorder sequence of this subtree suggests that 'B' is the root node of this left subtree. The position of 'B' in the inorder sequence determines the inorder sequence of the left and the right subtree as shown below:



And this process continues for the remaining set of subtrees, as follows.




A
B
C
D
E
F
G

Chapter-6 Sorting and Searching

Very Short Answers

Q.1

(i) Define searching.
Ans. Searching is a process in which an item is searched from a set of numbers.

(ii) Define sequential search.
Ans. The searching technioque in which the item is searched one by one until it is found or the list is exhausted is known as a sequential search.

(iii) State the condition under which Binary Search is applicable.
Ans. Binary Search is applicable on the set of numbers (list) which are either in ascending order or in descending order.

(iv) Define sorting.
Ans. Sorting is a technique in which all the items of a list are arranged either in ascending order or in descending order.

(v) Define recursion algorithm.
Ans. A recursive algorithm is one which calls itself directly or indirectly.

(vi) What is Radix sort?
Ans. Radix sort is a sorting algorithm that sorts integers by processing individual digits, by comparing individual digits sharing the same significant position.

(vii) Name two sorting algorithm.
Ans. Selection sorting and Bubble sorting

(viii) What do you mean by merge sort?
Ans. Merge sort is a process of combining two or more sorted lists into a third sorted list. Merge sort algorithm is based on 'divide-and-conquer' technique in which a list is divided into sublists.

(ix) Define heap sort.
Ans. The process of heap sorting is divided into two parts:

(i) In first part we create a heap H out of the elements of array
(ii) In second part we repeatedly delete the root element of H and move it to the last position of the array. After this we decrement the size of the array 'n', thereby excluding the largest value from further sorting.

Short Answer Type Questions

Q. 2 What is searching? Write the steps for linear search algorithm.

Ans. Identifying a particular item or a record among a set of elements or records is called as searching. Here is the algorithm of linear searching.

SequentialSearch(A, N, Item)
Here 'A' is an array of 'N' number of elements and 'Item' represents the item to be searched in array 'A'

1. Set Flag = 0

2. Initialize I = 0
3. Repeat through step-4 while (I < N)
4. Compare A[I] and item

5. Increment the value of 'I' as: I = I+1
6. If Flag = 1 then display the message – "Item Found";
   Otherwise display – "Item not found"
7. Exit

Q.3 Write down the C implementation of Binary Searching.

Ans.

```
Binarysearch(int A[], int n, int item)
{
        int low, high, mid, flag = 0;
        low = 0;
        high = n-1;
        while (low < high)
        {
                mid = (low+high)/2;
                if (A[mid] < item)
                low = mid + 1;
                else if (A[mid] > item )
                        high = mid - 1;
                else
                        flag = 1;
        }

}
```

Q.4 Explain insertion sorting with an example.

Ans. In an insertion sort, a set of elements is sorted by inserting elements into its proper position in the sorted list. Suppose 'A' is an array of 'N' numbers. The insertion sort techniques scans the array from A[0] to A[N-1], inserting each element into its proper position in the previously sorted subarray A[0], A[1], A[2], …., A[I-1]. Let us understand this insertion sorting by considering the following array:

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]
28	24	36	29	14	75	80	31	60	49

Then after each insertion we have the following array as shown in figure-6.5.

Passes	*pass-1*	*pass-2*	*pass-3*	*Pass-4*	*PASS-5*	*Pass-6*	*Pass-7*	*Pass-8*	*Pass-9*
A[0]	24	24	24	14	14	14	14	14	14
A[1]	28	28	28	24	24	24	24	24	24
A[2]	36	36	29	28	28	28	28	28	28
A[3]	29	29	36	29	29	29	29	29	29
A[4]	14	14	14	36	36	36	31	31	31
A[5]	75	75	75	75	75	75	36	36	36
A[6]	80	80	80	80	80	80	75	60	49
A[7]	31	31	31	31	31	31	80	75	60
A[8]	60	60	60	60	60	60	60	80	75
A[9]	49	49	49	49	49	49	49	49	80

Long Answer Type Questions

Q. 5. Explain binary search with suitable example.
Ans. Searching is a process in which an item is searched from a set of numbers. There are basically two searching techniques - Linear searching and Binary searching

Binary searching – In Binary Searching, searching begins from the middle of array. If the item is less than the item which is at the mid of array then it must be searched in the top half of array, that is 0 to mid-1. If the item is greater than the item which is at the mid of array then it must be searched in the second half of array that is mid+1 to n-1. If both are equal then no need to search further more. This process continues until the entire list is not further subdivided or item is found.

We perform binary searching by keeping two counter variables lower and upper. Two variables **lower** and **upper** are used to contain the lower value and upper value of the set of elements not yet searched. At each stage the number of elements in the remaining set is decreased by about one half. Initially we have following values:

```
lower = 0;
upper = n -1;
mid = (lower + upper)/2;
```

Then after first search if item is greater than a [mid] then

```
lower = mid+1 and
upper = n – 1
```

otherwise

```
lower = 0
upper = mid-1
```

Let we want to find a number 34 among the list of following numbers:

60, 53, 49, 40, 37, 34 and 21

Here are the steps to search 34 using Binary Search in the following array:

Number	60	53	49	40	37	34	21
Index	0	1	2	3	4	5	6

(i) Initially -
low = 0 and upper = 6
mid = (low + upper)/2 = (0+6)/2 = 3
a [mid] = a [3] = 40
Here item = 34 is less than a [mid], so we will update the value of lower to mid+1. Figure (a) shows this.

Number	60	53	49	40	37	34	21
Index	0	1	2	3	4	5	6
	Low			Mid			Upper

(a)

Number	60	53	49	40	37	34	21
Index	0	1	2	3	4	5	6
					Low	Mid	Upper

(b)
Figure - 2

(ii) lower = 4 and upper = 6
mid = (lower + upper)/2 =
(4+6)/2 mid = 5
a [mid] = a [5] = 34

Here item = 34 is equal to a [mid] as shown in figure 2(b), so the process of searching ends.

Q.6 Write program for binary search in C.
Ans.
```
#include <stdio.h>
#define Size 100
main()
{
	int a[Size];
	int i, n, item, flag;
	printf("\nEnter the size of array - ");
	scanf("%d", &n);
	printf("\nEnter the elements of array - \n");
	for(i=0; i<n; i++)
		scanf("%d", &a[i]);

	printf("\nEnter the element to be searched - ");
	scanf("%d", &item);
	flag = BinarySearch(a, n, item);

	if (flag == -1)
		printf("\nItem not found.");
	else
		printf("\nItem found at %dth position.", flag);
}
BinarySearch(int a[], int n, int item)
{
	int mid, lower, upper;
	lower=0;
	upper=n-1;
	while (lower <= upper)
	{
		mid = (lower+upper)/2;
		if (a[mid] == item)
			return (mid+1);
		else if (a[mid] > item)
			upper = mid-1;
		else
			lower = mid+1;
	}
	return (-1);
```

}

Q.7 Define Quick sort and implement it on the following array of elements

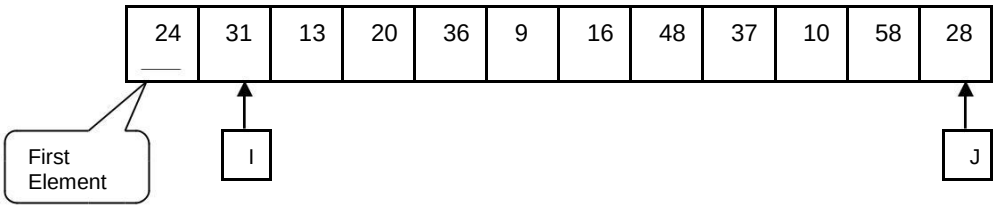
24	31	13	20	36	9	16	48	37	10	58	28

Ans. Quick sorting is a very fast method of internal sorting developed by *C.A.B. Hoare* in 1962. In this an element A_i is placed in such a way that all the elements from (A_0 A_1 A_2 A_3 ..... A_{i-1}) are less than A_i and all the elements from (A_{i+1}, A_{i+2}, A_{i+3}..... A_{n-1}) are equal to or greater than A_i. And this process is called recursively for (A_0 A_1 A_2 A_3 ..... A_{i-1}) and (A_{i+1}, A_{i+2}, A_{i+3}..... A_{n-1}) respectively. Each time the quick sorting function is invoked, it further divides the elements into smaller lists.

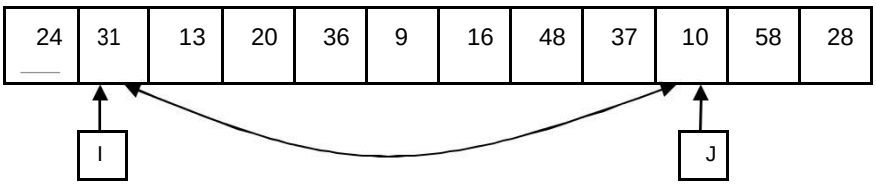
In Quick sorting, let A[First : Last] be an array to be sorted. Here we will assume two counter variables I and J in such a way that 'I' refers to 'First' and 'J' refers to 'Last'. Here 'First' indicates the lower bound of an array and 'Last' indicates upper bound of an array. The counter variable 'I' moves right searching for an element, which is greater than A[I] and the counter variable 'J' moves towards left for an element, which is smaller than A[I]. This process ends whenever the counter variable "I" and "J" meet or cross over. For example, consider an array of 10 elements as shown below:

Item	24	31	13	20	36	9	16	48	37	10	58	28
Index	0	1	2	3	4	5	6	7	8	9	10	11

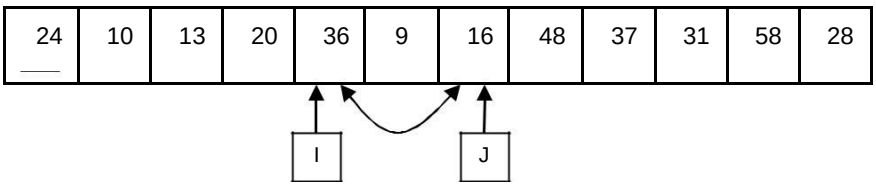
Here First = 0, Last = 11 and now we have to place the first element A[First] = 24 at its proper location. Initially 'I' is set to second and 'J' to last element of the array respectively as:



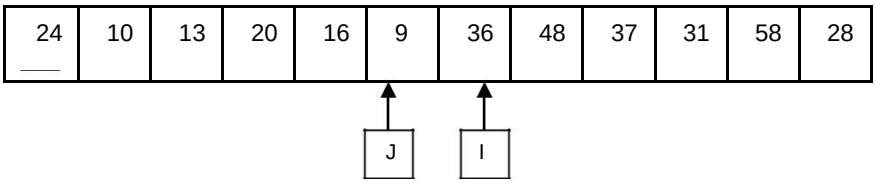
Now we move 'I' towards right until A[I] > 24. Since A[I] > 24, therefore 'I' does not move further to the right and stops immediately. Now we move 'J' towards left until A[J] < 22. Here 'J moves 2 units to the left and the scene may be as shown below:



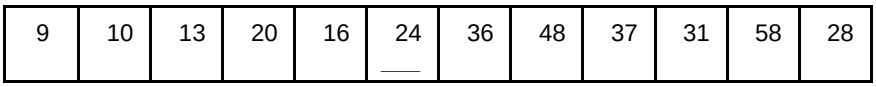
At this point, we exchange A[I] and A[J] and the movement of 'I' and 'J' resume. Now 'I' moves 3 units to the right and 'J' moves 3 units to left ass shown below:



Once again A[I] and A[J] are exchanged and movement of 'I' and 'J' resume.



Since the index of 'I' becomes greater than 'J', the process ends after exchanging A[J] and A[First]. The resultant array looks like the following:



Now the element 24 is placed at index '5'. This latest scenario of array 'A' clearly shows that elements of array A[ ], which are left to '5' are less than 24 and all elements which are right to index'5' are greater than 24. Thus the element 22 is placed correctly. Now the above procedure is applied on these two subarrays A[First : J-1] and A[J+1 : Last] and this process continues till the array id not further subdivided.

Here is the Quick sort algorithm:

QuickSort(A, First, Last)
Here 'A' is an array of elements and 'First' and 'Last' represent the first index and last index of the array 'A'.

Perform the following steps only if (I < J)
Step-1 : Set I = First+1;
Step-2 : Set J = Last
Step-3 : Repeat through step-8 while (I < J)
Step-4 : Repeat through step-5 while (A[I] < A[First])
Step-6 : Increment the value of 'I' as I = I+1
Step-6 : Repeat through step-7 while (A[J] > A[First])
Step-7 : Decrement the value of 'J' as J = J - 1
Step-8 : If (I < J) then exchange the value of A[I] and A[J]
Step-9 : Exchange the value of A[First] and A[j]
Step-10 : Call QuickSort (A, First, J-1);
Step-12 : Call QuickSort (A, J+1, Last);
Step-13 : Exit

Q. 8 Explain Bubble Sort technique? Implement it on the following array

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]
29	36	11	24	55	22

Or

Explain Bubble sort algorithm with example.

Ans. *Bubble sorting* compares the adjacent elements and moves the largest element to the bottom of the list. This process continues till the entire list is sorted.

Algorithm:

1. Iterates through every element of the array, starting with the first 2 elements.

2. If left element is bigger than right element, swap them.
3. Repeat step #1 and #2 until there are no more swaps.

First Pass:

29	36	11	24	55	22	**no interchange**
29	36	11	24	55	22	*** Interchange these two number**
29	11	36	24	55	22	*** Interchange these two number**
29	11	24	36	55	22	**no interchange**
29	11	24	36	55	22	*** Interchange these two number**
29	11	24	36	22	**55**	

Second Pass:

29	11	24	36	22	**55**	*** Interchange these two number**
11	29	24	36	22	**55**	*** Interchange these two number**
11	24	29	36	22	**55**	**no interchange**
11	24	29	36	22	55	*** Interchange these two number**
11	24	29	22	**36**	**55**	

Third Pass:

11	24	29	22	**36**	**55**	**no interchange**
11	24	29	22	**36**	**55**	**no interchange**
11	24	29	22	**36**	**55**	*** Interchange these two number**
11	24	22	**29**	**36**	**55**	

Fourth Pass

11	24	22	**29**	**36**	**55**	**no interchange**
11	24	22	**29**	**36**	**55**	*** Interchange these two number**
11	22	**24**	**29**	**36**	**55**	

Fifth Pass

11 22 **24 29 36 55 no interchange**

Final Sorted List:

11 22 24 29 36 55

Q.9 Following numbers

89, 20, 31, 56, 25, 64, 48

are to be sorted using selection sort. Show the list appear at the end of each pass.

Ans.

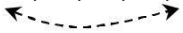
First Pass:

89, 20, 31, 56, 25, 64, 48 * Interchange these two number

20, 89, 31, 56, 25, 64, 48 no interchange

20, 89, 31, 56, 25, 64, 48 no interchange

20, 89, 31, 56, 25, 64, 48 no interchange



20, 89, 31, 56, 25, 64, 48 no interchange

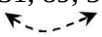
20, 89, 31, 56, 25, 64, 48 no interchange

Second Pass:

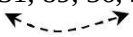
89, 20, 31, 56, 25, 64, 48 * Interchange these two number

20, 89, 31, 56, 25, 64, 48 no interchange

20, 31, 89, 56, 25, 64, 48 no interchange



20, 31, 89, 56, 25, 64, 48 * Interchange these two number



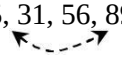
20, 25, 89, 56, 31, 64, 48 no interchange

20, 25, 89, 56, 31, 64, 48 no interchange

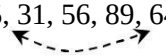
Third Pass:

20, 25, 89, 56, 31, 64, 48 * Interchange these two number

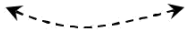
20, 25, 31, 56, 89, 64, 48 no interchange



20, 25, 31, 56, 89, 64, 48 no interchange



20, 25, 31, 56, 89, 64, 48 no interchange



Fourth Pass:

20, 25, 31, 56, 89, 64, 48	no interchange
20, 25, 31, 56, 89, 64, 48	no interchange
20, 25, 31, 56, 89, 64, 48	* Interchange these two number
20, 25, 31, 48, 89, 64, 56	

Fifth Pass:

20, 25, 31, 48, 89, 64, 56	* Interchange these two number
20, 25, 31, 48, 64, 89, 56	* Interchange these two number
20, 25, 31, 48, 56, 89, 64	

Sixth Pass:

20, 25, 31, 48, 56, 89, 64	* Interchange these two number
20, 25, 31, 48, 56, 64, 56	

Final Sorted List:

20, 25, 31, 48, 56, 64, 56

Q.10 Using merge sort technique sort an array containing elements:

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

Ans. Merging is a process of combining two or more sorted lists into a third sorted list. Merge sort algorithm is based on 'divide-and-conquer' technique in which a list is divided into sublists. This technique may be used effectively sort a list of numbers. In merge sorting, an unsorted array A[lower : upper] is split around its middle element , mid such that mid = (lower + upper) / 2, into two unsorted arrays A[lower : mid] and A[mid+1 : upper]. The same procedure is applied recursively on two unsorted arrays A[lower : mid] and A[mid+1 : upper] and finally when there are only one element left in the array, they are merged.

Now let us use merge sorting on following elements:

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

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

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

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

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

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

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

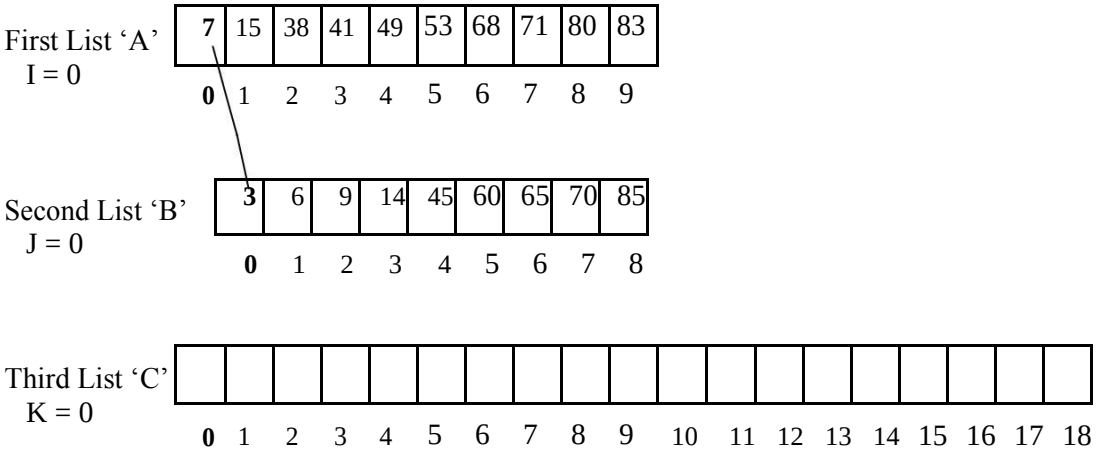
(11, 20, 22, 30, 33, 40, 44, 50, 55, 60, 66, 77, 80, 88)

Q.11 Write Merge sort algorithm. Describe the steps to sort the following list of elements using merge sort.

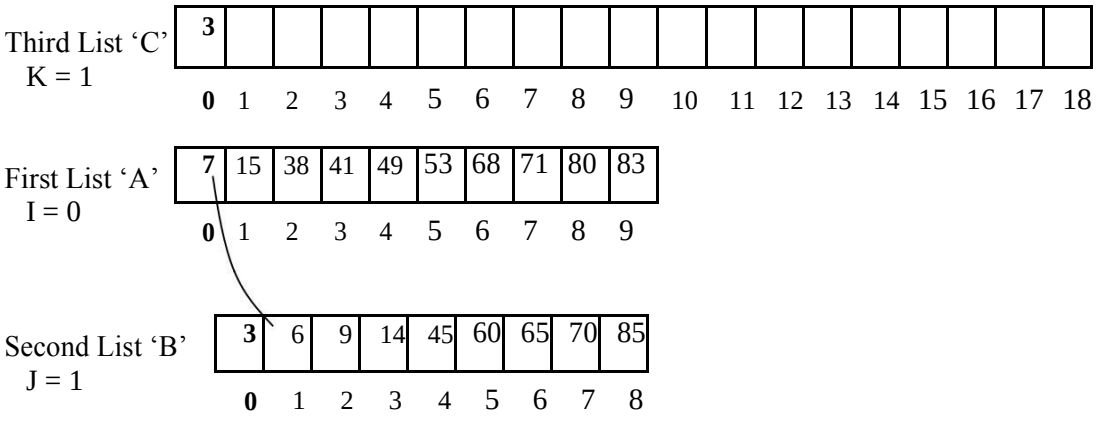
First List: 7, 15, 38, 41, 49, 53, 68, 71, 82, 90
Second List: 3, 6, 9, 14, 45, 60, 65, 70, 85

Ans. Let the name of first list is A and the second list is B. For list 'A', we consider the counter variable 'I' and for list 'B' we take 'J'. When we merge these two lists we will store it into a third list 'C'. For list 'C' we take the counter variable 'K'. Initially all these three counter variables are set to 0.

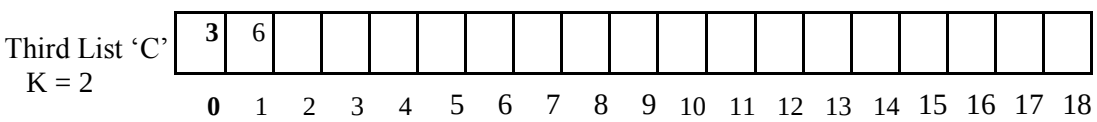
<u>Rule:</u> Both A[I] and B[J] are compared and smallest of these two is stored in third list and their respective counter variables are incremented.

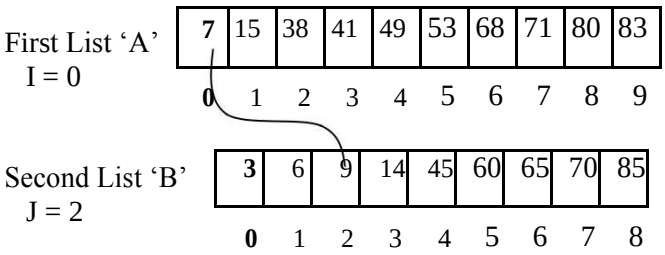


Here B[J] is smaller than A[I] therefore B[J] is stored at C[K] and both 'J' and 'K' are incremented as:

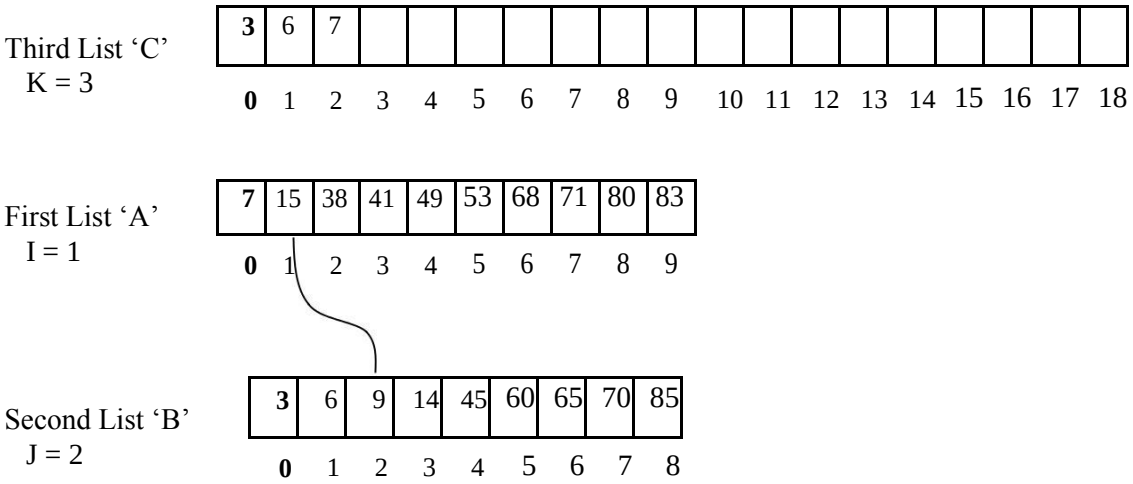


Again B[J] is smaller than A[I] therefore B[J] is stored at C[K] and both 'J' and 'K' are incremented as:

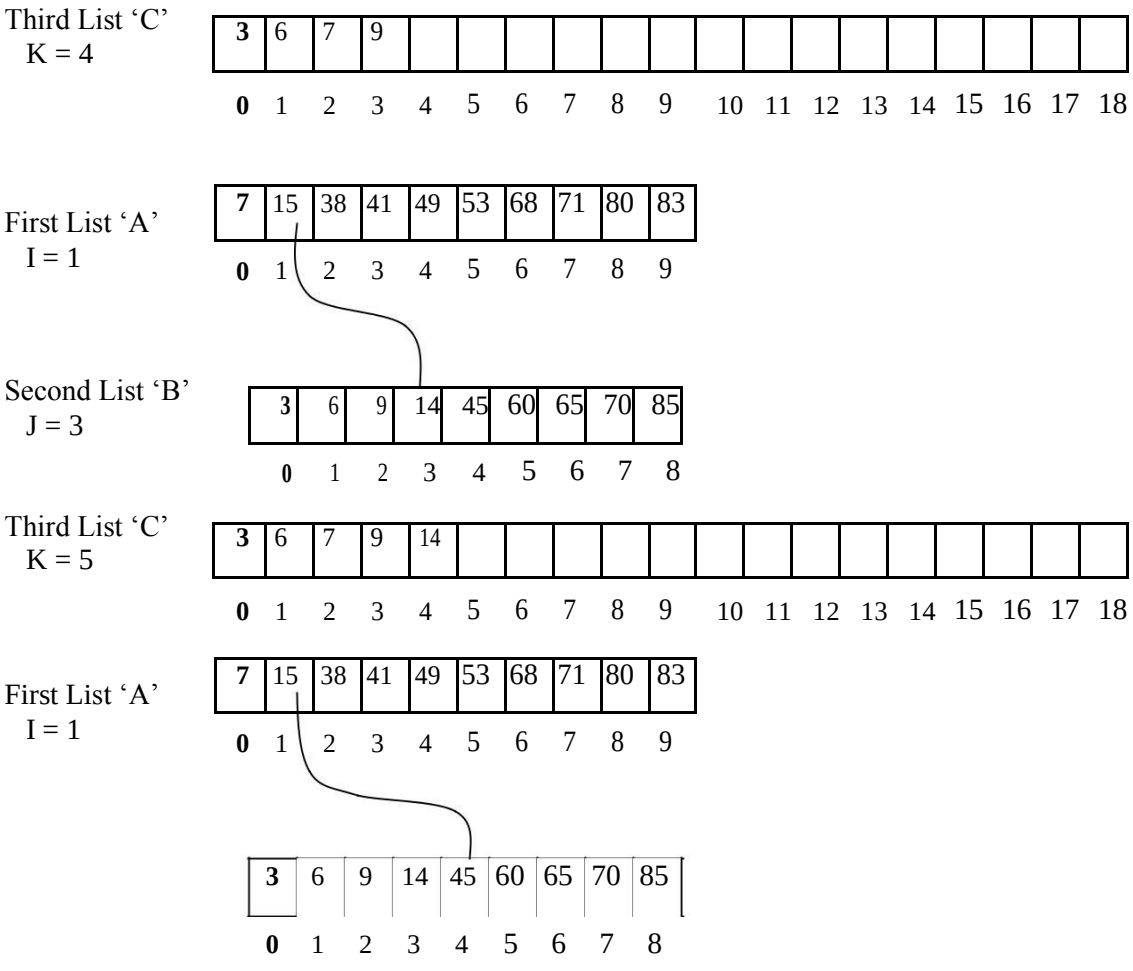




In third case B[J] is larger than A[I] therefore A[I] is stored at C[K] and both ‘I’ and ‘K’ are incremented as:

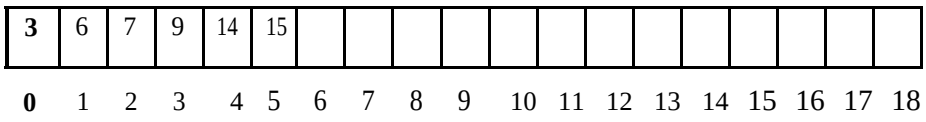


Next B[J] is smaller than A[I] therefore B[J] is stored at C[K] and both ‘J’ and ‘K’ are incremented as:

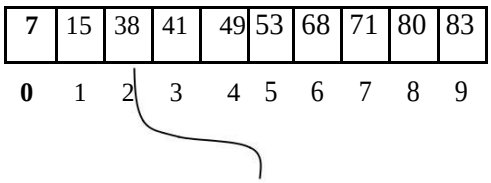


Second List ‘B’
J = 4

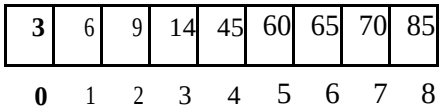
Third List ‘C’
K = 6



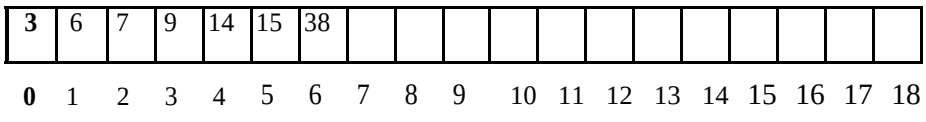
First List ‘A’
I = 2



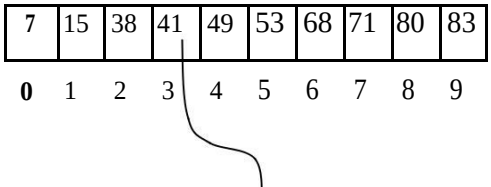
Second List ‘B’
J = 4



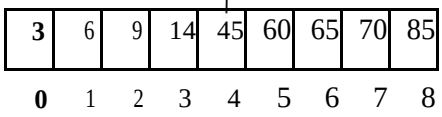
Third List ‘C’
K = 7



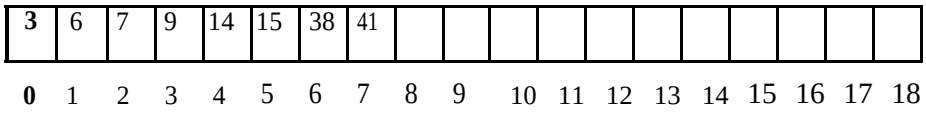
First List ‘A’
I = 3



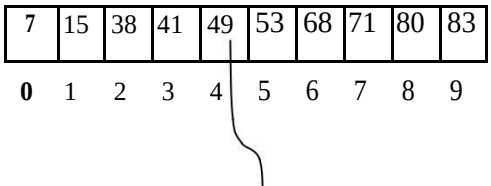
Second List ‘B’
J = 4



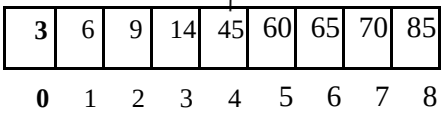
Third List ‘C’
K = 8



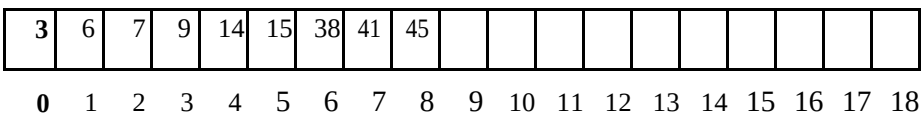
First List ‘A’
I = 4

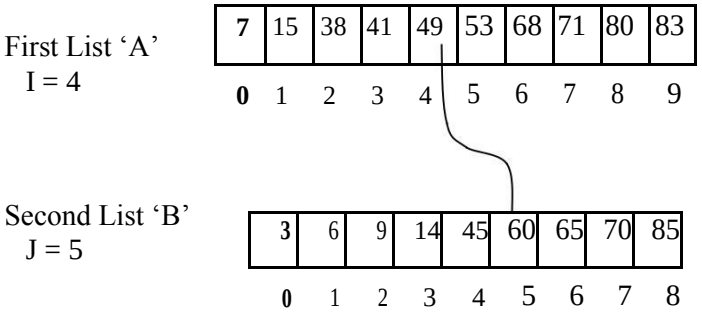


Second List ‘B’
J = 4



Third List ‘C’
K = 9





And this process continues and finally we get the sorted merged list as: