Maharaja Education Trust (R), Mysuru

Maharaja Institute of Technology Mysore

Belawadi, Sriranga Pattana Taluk, Mandya – 571 477

Approved by AICTE, New Delhi,
Affiliated to VTU, Belagavi & Recognized by Government of Karnataka

Lecture Notes on

Data Structures Laboratory (18CSL38)

Prepared by,

Kavya Priya M L,

Assistant Professor, CSE, MITM

Department of Computer Science & Engineering

Maharaja Education Trust (R), Mysuru
Maharaja Institute of Technology Mysore
Belawadi, Sriranga Pattana Taluk, Mandya – 571 477

Vision/ ಆಶಯ

"To be recognized as a premier technical and management institution promoting extensive education fostering research, innovation and entrepreneurial attitude"

ಸಂಶೋಧನೆ, ಆವಿಷ್ಕಾರ ಹಾಗೂ ಉದ್ಯಮಶೀಲತೆಯನ್ನು ಉತ್ತೇಜಿಸುವ ಅಗ್ರಮಾನ್ಯ ತಾಂತ್ರಿಕ ಮತ್ತು ಆಡಳಿತ ವಿಜ್ಞಾನ ಶಿಕ್ಷಣ ಕೇಂದ್ರವಾಗಿ ಗುರುತಿಸಿಕೊಳ್ಳುವುದು.

Mission/ ಧ್ಯೇಯ

- To empower students with indispensable knowledge through dedicated teaching and collaborative learning.

  ಸಮರ್ಪಣಾ ಮನೋಭಾವದ ಬೋಧನೆ ಹಾಗೂ ಸಹಭಾಗಿತ್ವದ ಕಲಿಕಾಕ್ರಮಗಳಿಂದ ವಿದ್ಯಾರ್ಥಿಗಳನ್ನು ಅತ್ಯತ್ಕೃಷ್ಟ ಜ್ಞಾನಸಂಪನ್ನರಾಗಿಸುವುದು.

- To advance extensive research in science, engineering and management disciplines.

  ವೈಜ್ಞಾನಿಕ, ತಾಂತ್ರಿಕ ಹಾಗೂ ಆಡಳಿತ ವಿಜ್ಞಾನ ವಿಭಾಗಗಳಲ್ಲಿ ವಿಸ್ತೃತ ಸಂಶೋಧನೆಗಳೊಡನೆ ಬೆಳವಣಿಗೆ ಹೊಂದುವುದು.

- To facilitate entrepreneurial skills through effective institute - industry collaboration and interaction with alumni.

  ಉದ್ಯಮ ಕ್ಷೇತಗಳೊಡನೆ ಸಹಯೋಗ, ಸಂಸ್ಥೆಯ ಹಿರಿಯ ವಿದ್ಯಾರ್ಥಿಗಳೊಂದಿಗೆ ನಿರಂತರ ಸಂವಹನಗಳಿಂದ ವಿದ್ಯಾರ್ಥಿಗಳಿಗೆ ಉದ್ಯಮಶೀಲತೆಯ ಕೌಶಲ್ಯ ಪಡೆಯಲು ನೆರವಾಗುವುದು.

- To instill the need to uphold ethics in every aspect.

  ಜೀವನದಲ್ಲಿ ನೈತಿಕ ಮೌಲ್ಯಗಳನ್ನು ಅಳವಡಿಸಿಕೊಳ್ಳುವುದರ ಮಹತ್ವದ ಕುರಿತು ಅರಿವು ಮೂಡಿಸುವುದು.

- To mould holistic individuals capable of contributing to the advancement of the society.

  ಸಮಾಜದ ಬೆಳವಣಿಗೆಗೆ ಗಣನೀಯ ಕೊಡುಗೆ ನೀಡಬಲ್ಲ ಪರಿಪೂರ್ಣ ವ್ಯಕ್ತಿತ್ವವುಳ್ಳ ಸಮರ್ಥ ನಾಗರೀಕರನ್ನು ರೂಪಿಸುವುದು.

Maharaja Institute of Technology Mysore

Department of Computer Science & Engineering

Vision / ಆಶಯ

To be recognized as numero uno in the field of civil engineering education, research and an imparter of enterprising skills.

ಸಿವಿಲ್ ಇಂಜಿನಿಯರಿಂಗ್ ಶಿಕ್ಷಣ, ಸಂಶೋಧನೆ ಹಾಗೂ ಉದ್ಯಮಶೀಲತೆಯ ಕೌಶಲ್ಯಗಳನ್ನು ಒದಗಿಸುವ ಅಗ್ರಮಾನ್ಯ ವಿಭಾಗವಾಗಿ ಗುರುತಿಸಿಕೊಳ್ಳುವುದು.

Mission / ಧ್ಯೇಯ

- To facilitate technical ingenuity through proficient teaching learning processes that inspires self learning.

  ಬೋಧನಾ ಪ್ರಾವೀಣ್ಯತೆ ಹಾಗೂ ಕಲಿಕಾಕ್ರಮಗಳ ಮೂಲಕ ಸ್ವಯಂ ಕಲಿಕೆಯನ್ನು ಉತ್ತೇಜಿಸುವ ತಾಂತ್ರಿಕ ಚಾತುರ್ಯವನ್ನು ಪಡೆಯಲು ನೆರವಾಗುವುದು.

- To enhance collaborative growth in research and consultancy and deliver solutions to meet the societal needs.

  ಸಾಮಾಜಿಕ ಅಗತ್ಯತೆಗಳಿಗೆ ತಾಂತ್ರಿಕ ಪರಿಹಾರ ಕಲ್ಪಿಸಬಲ್ಲ ಸಂಶೋಧನೆ ಹಾಗೂ ತಜ್ಞ ಸಲಹೆಗಾಗಿ ಸಹಭಾಗಿತ್ವದ ಬೆಳವಣಿಗೆಯನ್ನು ವೃದ್ಧಿಸುವುದು.

- To instill students with communication and professional skills to foster industry-academia interaction that propels entrepreneurial attitude.

  ಉದ್ಯಮಶೀಲತೆಯ ಮನೋವೃತ್ತಿಯನ್ನು ಮುನ್ನೆಲೆಗೆ ತರಬಲ್ಲ ಉದ್ದಿಮೆ - ಶಿಕ್ಷಣ ವಲಯಗಳ ನಡುವಿನ ಬಾಂಧವ್ಯವರ್ಧನೆಯಲ್ಲಿ ಪಾಲ್ಗೊಳ್ಳಲು ಅಗತ್ಯವಿರುವ ಸಂವಹನ ಹಾಗೂ ವೃತ್ತಿಪರ ಕೌಶಲ್ಯಗಳನ್ನು ವಿದ್ಯಾರ್ಥಿಗಳಲ್ಲಿ ಹುಟ್ಟು ಹಾಕುವುದು.

Program Outcomes

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Maharaja Institute of Technology Mysore

Department of Computer Science and Engineering

Course Overview

SUBJECT: DATA STRUCTURES LABORATORY **SUBJECT CODE: 18CSL38**

In computer science, a data structure is a data organization, management, and storage format that enables efficient access and modification. Data structures are generally based on the ability of a computer to fetch and store data at any place in its memory, specified by a pointer-a bit string, representing a memory address, that can be itself stored in memory and manipulated by the program. Thus, the array and record data structures are based on computing the addresses of data items with arithmetic operations, while the linked data structures are based on storing addresses of data items within the structure itself.

The implementation of a data structure usually requires writing a set of procedures that create and manipulate instances of that structure. The efficiency of a data structure cannot be analysed separately from those operations. This observation motivates the theoretical concept of an abstract data type, a data structure that is defined indirectly by the operations that may be performed on it, and the mathematical properties of those operations. Different types of data structures are suited to different kinds of applications, and some are highly specialized to specific tasks. For example, relational databases commonly use B-tree indexes for data retrieval, while compiler implementations usually use hash tables to look up identifiers.

Data structures provide a means to manage large amounts of data efficiently for uses such as large databases and internet indexing services. Usually, efficient data structures are key to designing efficient algorithms. Some formal design methods and programming languages emphasize data structures, rather than algorithms, as the key organizing factor in software design. Data structures can be used to organize the storage and retrieval of information stored in both main memory and secondary memory.

Course Objectives

- Explain fundamentals of data structures and their applications essential for programming/problem solving.
- Illustrate linear representation of data structures: Stack, Queues, Lists, Trees and Graphs.
- Demonstrate sorting and searching algorithms.
- Find suitable data structure during application development/Problem Solving.

Course Outcomes

CO's	DESCRIPTION OF THE OUTCOMES
18CSL38.1	Apply the concept of data structure through ADT including Stacks, Queues, Lists.
18CSL38.2	Implement appropriate sorting, searching and traversing technique for given problem.
18CSL38.3	Implement a suitable solution for solving problems on non-linear data structure.
18CSL38.4	Analyse the problems with respect to data structures and document the result in a required format.

Maharaja Institute of Technology Mysore

Department of Computer Science and Engineering

Syllabus

SUBJECT: DATA STRUCTURES LABORATORY **SUBJECT CODE:18CSL38**

Topics Covered as per Syllabus
1. Design, Develop and Implement a menu driven Program in C for the following **Array** operations a. Creating an Array of **N** Integer Elements b. Display of Array Elements with Suitable Headings c. Inserting an Element (**ELEM**) at a given valid Position (**POS**) d. Deleting an Element at a given valid Position(**POS**) e. Exit. Support the program with functions for each of the above operations. 2. Design, Develop and Implement a Program in C for the following operations on **Strings** a. Read a main String (**STR),** a Pattern String (**PAT**) and a Replace String (**REP**) b. Perform Pattern Matching Operation: Find and Replace all occurrences of **PAT** in **STR** with **REP** if **PAT** exists in **STR.** Report suitable messages in case **PAT** does not exist in **STR** Support the program with functions for each of the above operations. Don't use Built-in functions. 3. Design, Develop and Implement a menu driven Program in C for the following operations on **STACK** of Integers (Array Implementation of Stack with maximum size **MAX**) a. ***Push*** an Element on to Stack b. ***Pop*** an Element from Stack *c.* Demonstrate how Stack can be used to check ***Palindrome*** d. Demonstrate ***Overflow*** and ***Underflow*** situations on Stack e. Display the status of Stack f. Exit Support the program with appropriate functions for each of the above operations 4. Design, Develop and Implement a Program in C for converting an Infix Expression to Postfix Expression. Program should support for both parenthesized and free parenthesized expressions with the operators: **+, -, *, /, %(Remainder), ^(Power)** and **alphanumeric** operands**.** 5. Design, Develop and Implement a Program in C for the following Stack Applications a. Evaluation of **Suffix expression** with single digit operands and operators: **+, -, *, /, %,** ^ b. Solving **Tower of Hanoi** problem with **n** disks 6. Design, Develop and Implement a menu driven Program in C for the following operations on **Circular QUEUE** of Characters (Array Implementation of Queue with maximum size **MAX**)

a. Insert an Element on to Circular QUEUE
b. Delete an Element from Circular QUEUE
c. Demonstrate ***Overflow*** and ***Underflow*** situations on Circular QUEUE
d. Display the status of Circular QUEUE
e. Exit
Support the program with appropriate functions for each of the above operations

7. Design, Develop and Implement a menu driven Program in C for the following operations on **Singly Linked List (SLL)** of Student Data with the fields: ***USN, Name, Branch, Sem, PhNo***
a. Create a **SLL** of **N** Students Data by using ***front insertion***.
b. Display the status of **SLL** and count the number of nodes in it
c. Perform Insertion / Deletion at End of **SLL**
d. Perform Insertion / Deletion at Front of **SLL(Demonstration of stack)**
e. Exit

8. Design, Develop and Implement a menu driven Program in C for the following operations on **Doubly Linked List (DLL)** of Employee Data with the fields: ***SSN, Name, Dept, Designation, Sal, PhNo***
a. Create a **DLL** of **N** Employees Data by using ***end insertion***.
b. Display the status of **DLL** and count the number of nodes in it
c. Perform Insertion and Deletion at End of **DLL**
d. Perform Insertion and Deletion at Front of **DLL**
e. Demonstrate how this **DLL** can be used as **Double Ended Queue**
f. Exit

9. Design, Develop and Implement a Program in C for the following operations on **Singly Circular Linked List (SCLL)** with header nodes
a. Represent and Evaluate a Polynomial **P(x,y,z) = 6x2y2z-4yz5+3x3yz+2xy5z- 2xyz3**
b. Find the sum of two polynomials **POLY1(x,y,z)** and **POLY2(x,y,z)** and store the result in **POLYSUM(x,y,z)**
Support the program with appropriate functions for each of the above operations

10. Design, Develop and Implement a menu driven Program in C for the following operations on **Binary Search Tree (BST)** of Integers
a. Create a BST of **N** Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2
b. Traverse the BST in Inorder, Preorder and Post Order
c. Search the BST for a given element (**KEY**) and report the appropriate message
e. Exit

11. Design, Develop and Implement a Program in C for the following operations on **Graph(G)** of Cities
a. Create a Graph of **N** cities using Adjacency Matrix.
b. Print all the nodes **reachable** from a given starting node in a digraph using DFS/BFS method

12. Given a File of **N** employee records with a set **K** of Keys(4-digit) which uniquely determine the records in file **F**. Assume that file **F** is maintained in memory by a Hash Table(HT) of **m** memory locations with **L** as the set of memory addresses (2- digit) of locations in HT. Let the keys in **K** and addresses in **L** are Integers. Design and develop a Program in C that uses Hash function **H: K L** as H(**K**)=**K** mod **m (remainder** method), and implement hashing technique to map a given key **K** to the address space **L.** Resolve the collision (if any) using **linear probing**.

Conduct of Practical Examination:

- Experiment distribution

o For laboratories having only one part: Students are allowed to pick one experiment from the lot with equal opportunity.

o For laboratories having PART A and PART B: Students are allowed to pick one experiment from PART A and one experiment from PART B, with equal opportunity.

- Change of experiment is allowed only once and marks allotted for procedure to be made zero of the changed part only.
- Marks Distribution *(Courseed to change in accoradance with university regulations)*

c) For laboratories having only one part – Procedure + Execution + Viva-Voce: 15+70+15 = 100 Marks

d) For laboratories having PART A and PART B

i. Part A – Procedure + Execution + Viva = 6 +28 + 6 = 40 Marks

ii. Part B – Procedure + Execution + Viva = 9 + 42 + 9 = 60 Marks

Index

SUBJECT: DATA STRUCTURES LABORATORY **SUBJECT CODE: 18CSL38**

LABORATORY

General Lab Guidelines:

- Conduct yourself in a responsible manner at all times in the laboratory. Intentional misconduct will lead to the exclusion from the lab.
- Do not wander around, or distract other students, or interfere with the laboratory experiments of other students.
- Read the handout and procedures before starting the experiments. Follow all written and verbal instructions carefully. If you do not understand the procedures, ask the instructor or teaching assistant.
- Attendance in all the labs is mandatory, absence permitted only with prior permission from Class teacher.
- The workplace has to be tidy before, during and after the experiment.
- Do not eat food, drink beverages or chew gum in the laboratory.
- Every student should know the location and operating procedures of all Safety equipment including First Aid Kit and Fire extinguisher.

DO'S:-

- Uniform and ID card are must.
- Records have to be submitted every week for evaluation.
- Sign the log book when you enter/leave the laboratory.
- After the lab session, shut down the computers.
- Keep your belongings in designated area.
- Report any problem in system (if any) to the person in-charge.

DONT'S:-

- Do not insert metal objects such as clips, pins and needles into the computer casings(They may cause fire) and should not attempt to repair, open, tamper or interfere with any of the computer, printing, cabling, or other equipment in the laboratory.
- Do not change the system settings and keyboard keys.
- Do not upload, delete or alter any software/ system files on laboratory computers.
- No additional material should be carried by the students during regular labs.
- Do not open any irrelevant websites in labs.
- Do not use a flash drive on lab computers without the consent of lab instructor.
- Students are not allowed to work in Laboratory alone or without presence of the instructor/teaching assistant.

DATA STRUCTURES

Data Structures:

The logical or mathematical model of a particular organization of data is called data structures. Data structures is the study of logical relationship existing between individual data elements, the way the data is organized in the memory and the efficient way of storing, accessing and manipulating the data elements.

Choice of a particular data model depends on two considerations: it must be rich enough in structure to mirror the actual relationships of the data in the real world. On the other hand, the structure should be simple enough that one can effectively process the data when necessary.

Data Structures can be classified as:
Primitive data structures
Non-Primitive data structures.

Primitive data structures are the basic data structures that can be directly manipulated/operated by machine instructions. Some of these are character, integer, real, pointers etc.

Non-primitive data structures are derived from primitive data structures, they cannot be directly manipulated/operated by machine instructions, and these are group of homogeneous or heterogeneous data items. Some of these are Arrays, stacks, queues, trees, graphs etc.

Data structures are also classified as
Linear data structures
Non-Linear data structures.

In the Linear data structures processing of data items is possible in linear fashion, i.e., data can be processed one by one sequentially.
Example of such data structures are:
Array
Linked list
Stacks
Queues

A data structure in which insertion and deletion is not possible in a linear fashion is called as non linear data structure. i.e., which does not show the relationship of logical adjacency between the elements is called as non-linear data structure. Such as trees, graphs and files.

Data structure operations:

The particular data structures that one chooses for a given situation depends largely on the frequency with which specific operations are performed. The following operations play major role in the processing of data.

i) Traversing.
ii) Searching.
iii) Inserting.
iv) Deleting.

v) Sorting.

vi) Merging

STACKS:

A stack is an ordered collection of items into which new items may be inserted and from which items may be deleted at the same end, called the TOP of the stack. A stack is a non-primitive linear data structure. As all the insertion and deletion are done from the same end, the first element inserted into the stack is the last element deleted from the stack and the last element inserted into the stack is the first element to be deleted. Therefore, the stack is called Last-In First-Out (LIFO) data structure.

QUEUES:

A queue is a non-primitive linear data structure. Where the operation on the queue is based on First-In-First-Out FIFO process — the first element in the queue will be the first one out. This is equivalent to the requirement that whenever an element is added, all elements that were added before have to be removed before the new element can be removed.

For inserting elements into the queue are done from the rear end and deletion is done from the front end, we use external pointers called as rear and front to keep track of the status of the queue. During insertion, Queue Overflow condition has to be checked. Likewise during deletion, Queue Underflow condition is checked.

LINKED LIST:

Disadvantages of static/sequential allocation technique:

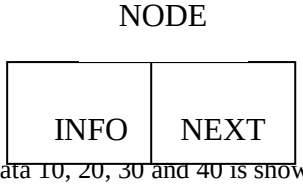
1) If an item has to be deleted then all the following items will have to be moved by one allocation. Wastage of time.
2) Inefficient memory utilization.
3) If no consecutive memory (free) is available, execution is not possible.

Linear Linked Lists
Types of Linked lists:
1) Single Linked lists
2) Circular Single Linked Lists
3) Double Linked Lists
4) Circular Double Linked Lists.

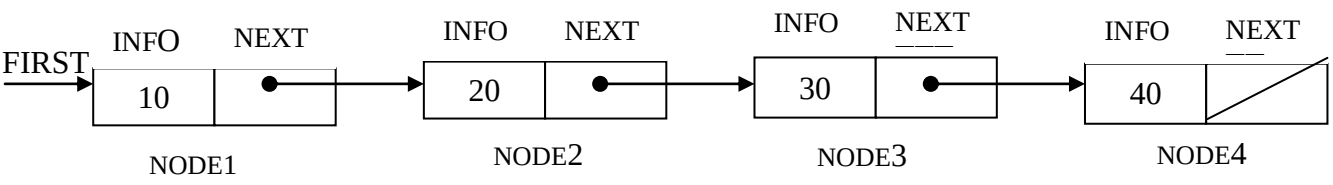
NODE:

Each node consists of two fields. Information (info) field and next address(next) field. The info field consists of actual information/data/item that has to be stored in a list. The second field next/link contains the address of the next node. Since next field contains the address, It is of type pointer.Here the nodes in the list are logically adjacent to each other. Nodes that are physically adjacent need not be logically adjacent in the list.

The entire linked list is accessed from an external pointer FIRST that points to (contains the address of) the first node in the list. (By an "external" pointer, we mean, one that is not included within a node. Rather its value can be accessed directly by referencing a variable).



The list containing 4 items/data 10, 20, 30 and 40 is shown below.



The nodes in the list can be accessed using a pointer variable. In the above fig. FIRST is the pointer having the address of the first node of the list, initially before creating the list, as list is empty. The FIRST will always be initialized to NULL in the beginning. Once the list is created, FIRST contains the address of the first node of the list.
As each node is having only one link/next, the list is called single linked list and all the nodes are linked in one direction.

Each node can be accessed by the pointer pointing (holding the address) to that node, Say P is pointer to a particular node, then the information field of that node can be accessed using info(P) and the next field can be accessed using next(P).
The arrows coming out of the next field in the fig. indicates that the address of the succeeding node is stored in that field.
The link field of last node contains a special value known as NULL which is shown using a diagonal line pictorially. This NULL pointer is used to signal the end of a list.

The basic operations of linked lists are Insertion, Deletion and Display. A list is a dynamic data structure. The number of nodes on a list may vary dramatically as elements are inserted and deleted(removed). The dynamic nature of list may be contrasted with the static nature of an array, whose size remains constant. When an item has to inserted, we will have to create a node, which has to be got from the available free memory of the computer system, So we shall use a mechanism to find an unused node which makes it available to us. For this purpose we shall use the getnode operation (getnode() function).

The C language provides the built-in functions like malloc(), calloc(), realloc() and free(), which are stored in alloc.h or stdlib.h header files. To dynamically allocate and release the memory locations from/to the computer system.

TREES:

Definition: A data structure which is accessed beginning at the root node. Each node is either a leaf or an internal node. An internal node has one or more child nodes and is called the parent of its child nodes. All children of the same node are siblings. Contrary to a physical tree, the root is usually depicted at the top of the structure, and the leaves are depicted at the bottom. A tree can also be defined as a connected, acyclic di-graph.

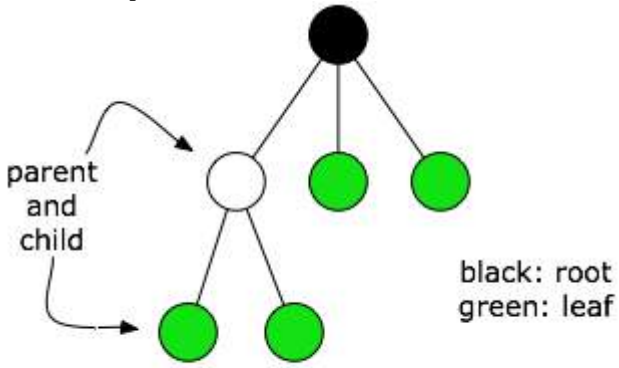


Figure: tree data structure

Binary tree: A tree with utmost two children for each node. Complete binary tree: A binary tree in which every level, except possibly the deepest, is completely filled. At depth n, the height of the tree, all nodes must be as far left as possible. Binary search tree: A binary tree where every node's left subtree has keys less than the node's key, and every right subtree has keys greater than the node's key. Tree traversal is a technique for processing the nodesof a tree in some order. The different tree traversal techniques are Pre-order, In-order and Post-order traversal. In Pre-order traversal, the tree node is visited first and the left subtree is traversed recursively and later right sub-tree is traversed recursively.

Graph:

Definition: A Graph G consists of two sets, V and E. V is a finite set of vertices. E is a set of edges or pair of vertices.

Two types of graphs, Undirected Graph Directed Graph

Undirected Graph: In undirected graph the pair of vertices representing any edge is unordered. Thus the pairs (u,v)and (v,u) represent the same edge.

Directed graph: In a directed graph each edge is represented by a directed pair <u,v>, u is the tail and v is the head of the edge. Therefore <v,u>and <u,v> represent two different edges.

Graph representation can be done using adjacency matrix.

Adjacency matrix: Adjacency matrix is a two dimensional n x n array a, with the property that a[i][j]=1 iff the edge (i,j) is in E(G). a[i][j]=0 if there is no such edge in G.

Connectivity of the graph: A graph is said to be connected iff for every pair of distinct vertices u and v , there is a path from u to v.

Path: A path from vertex u to v in a graph is a sequence of vertices u, i_1 , i_2 …… i_k , v such that (u, i_1), (i_1 , i_2) …… (i_k , v) are the edges in G.

Graph traversal can be done in two ways: depth first search(dfs) and breadth first search (bfs).

Hashing: Hashing is a process of generating key or keys from a string of text using a mathematical function called hash function. Hashing is a key-to-address mapping process.

Hash Table: In hashing the dictionary pairs are stored in a table called hash table. Hash tables are partitioned into buckets, buckets consists of s slots , each slot is capable of holding one dictionary pair.

Hash function: A hash function maps a key into a bucket in the hash table. Most commonly used hash function is,

$$h(k) = k \% D$$

where,
k is the key
D is Max size of hash table.

Collision Resolution: When we hash a new key to an address, collision may be created.
There are several methods to handle collision, Open addressing, linked lists and buckets.
In open addressing, several ways are listed, Linear probing, quadratic probe, pseudorandom, and key offset.

Linear Probing: In linear probing, when data cannot be stored at the home address,collision is resolved by adding 1 to the current address.

1. **Design, Develop and Implement a menu driven program in c for the following Array operations,**
   a. **Creating an Array of N integer elements.**
   b. **Display of Array elements with suitable headings.**
   c. **Inserting an element (ele) at a given valid position(pos).**
   d. **Deleting an element at a given valid position (pos).**
   e. **Exit.**

   Support the program with functions for each of the above operations.

```
#include<stdio.h>
#include<conio.h>
int n,a[50];

void create()
{
int i;
printf("enter the value of n\n");
scanf("%d",&n);
printf("enter %d array elements\n",n);
for(i=0;i<n;i++)
scanf("%d",&a[i]);
}

void display()
{
int i;
printf("entered elements are\n");
for(i=0;i<n;i++)
printf("%d\n",a[i]);
}

void insertion()
{
int i,POS,ELEM;
printf("enter the position and its value\n");
scanf("%d%d",&POS,&ELEM);
for(i=n;i>=POS;i--)
a[i]=a[i-1];
a[POS]=ELEM;
n=n+1;
display();
}

void deletion()
{
int i,POS,ELEM;
printf("enter the position to be deleted\n");
scanf("%d",&POS);
ELEM=a[POS];
for(i=POS;i<=n-1;i++)
a[i]=a[i+1];
printf("the deleated element is %d\n",ELEM);
n=n-1;
display();
}

void main()
{
int ch;
while(1)
{
```

```
printf("enter your choice\n");
printf("1.creat\n2.display\n3.insertion\n4.deletion\n5.exit\n");
scanf("%d",&ch);
switch(ch)
{
case 1: create();
break;
case 2: display();
break;
case 3: insertion();
break;
case 4: deletion();
break;
case 5: exit(0);
}
}
}
```

2. Design, Develop and Implement a program in c for the following operations on strings

a. Read a main string (str), a pattern string (pat) and a replace string (rep).

b. Perform pattern matching operation: Find and replace all occurrences of pat in str with rep if pat exits in str. Report with suitable messages in case pat does not exists in str.

Support the program with functions for each of the above operations. Don't use built in functions.

```
#include<stdio.h>
char STR[100],PAT[100],REP[100],ANS[100];
int i,j,c,m,k,flag=0;
void read()
{
printf("\n Enter MAIN string:\n");
gets(STR);
printf("\n Entre PATTERN string:\n");
gets(PAT);
printf("\n Enter REPLACE string\n");
gets(REP);
}

void replace()
{
 i=m=c=j=0;
 while(STR[c]!='\0')
 {
  if(STR[m]==PAT[i])
  {
   i++;m++;
    if(PAT[i]=='\0')
    {
     for(k=0;REP[k]!='\0';k++,j++)
     ANS[j]=REP[k];
     i=0;
     c=m;flag=1;
    }
   }
 else
  {
   ANS[j]=STR[c];
   j++;c++;m=c;i=0;
   }
 }
if(flag==0)
printf("pattern doesen't found!!!\n");
else
{
ANS[j]='\0';
printf("\n the RESULTANT string\n is %s \n",ANS);
}
}

void main()
{
clrscr();
read();
replace();
getch();
```

3. Design, Develop and Implement a menu driven program in c for the following

operations on STACK of integers(Array implementation of stack with maximum size MAX)

a. **Push an element onto the stack.**
b. **Pop an element from the stack.**
c. **Demonstrate how stack can be used to check palindrome.**
d. **Demonstrate overflow and underflow situations on stack.**
e. **Display the status of the stack.**
f. **Exit**

Support the program with appropriate functions for each of the above operations.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define max 5
int s[max],stop;
int ele,st[max],sp,ch;

void push(int ele,int s[],int *stop)
{
if(*stop>=max-1)
printf("stock overflow\n");
else
s[++*stop]=ele;
}

int pop(int s[],int *top)
{
if(*top==-1)
{
printf("stock empty\underflow\n");
return 0;
}
else
return(s[(*top)--]);
}

void palindrome(int ele,int st[])
{
int rem,rev=0,temp=ele,i=0;
while(temp!=0)
{
rem=temp%10;push(rem,st,&sp);
temp=temp/10;
}
while(sp!=-1)
rev=rev+(pop(st,&sp)*pow(10,i++));
if(ele==rev)
printf("palendrome\n");
else
printf("not a palindrome\n");
}

void display(int s[],int *stop)
{
int i;
if(*stop==-1)
printf("stalk is empty\n");
else
for(i=*stop;i>-1;i--)
printf("%d\n",s[i]);
```

```
}

void main()
{
stop= -1;
sp= -1;

while(1)
{
printf("enter tne choice\n");
printf("enter 1 to insert an element into the STACK\n");
printf("enter 2 to delete an element from the STACK\n");
printf("enter 3 to check an element is palindrome or not\n");
printf("enter 4 to check the status of the STACK\n");
printf("enter 5 to exit\n");
scanf("%d",&ch);
switch(ch)
{
case 1:printf("enter the elements to de inserted to STACK\n");
      scanf("%d",&ele);
      push(ele,s,&stop);
      break;
case 2:ele=pop(s,&stop);
      if(ele!=0)
      printf("element poped is %d\n",ele);
      break;
case 3:printf("enter the elements to chech weather it is a palindrome\n");
      scanf("%d",&ele);
      palindrome(ele,st);
      break;
case 4:printf("the status of the STACK \n");
      display(s,&stop);
      break;
case 5:exit(0);
}
}
}
```

4. Design, develop, and implement a program in C for converting an infix expression to postfix expression. Program should support for both parenthesized and free parenthesized expressions with the operators: +, - , *,/,% (Remainder),^(power) and alphanumeric operands.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int F(char symbol)
{
  switch(symbol)
  {
            case '+':
            case '-':return 2;
            case '*':
            case '%':
            case '/':return 4;
            case '^':
            case '$':return 5;
            case '(':return 0;
            case '#':return -1;
            default :return 8;
  }
}
int G(char symbol)
{
            switch(symbol)
            {
            case '+':
            case '-':return 1;
            case '*':
            case '%':
            case '/':return 3;
            case '^':
            case '$':return 6;
            case '(':return 9;
            case ')':return 0;
            default :return 7;
            }
}
void infix_postfix(char infix[],char postfix[])
{
 int top,i,j;
 char s[30];
 char symbol;
 top=-1;
 s[++top]='#';
 j=0;
 for(i=0;i<strlen(infix);i++)
 {
  symbol=infix[i];
  while(F(s[top])>G(symbol))
   postfix[j++]=s[top--];

   if(F(s[top])!=G(symbol))
      s[++top]=symbol;
   else top--;
 }

 while(s[top]!='#')
  postfix[j++]=s[top--];
```

```
  postfix[j]='\0';
}

void main()
{
char infix[20];
char postfix[20];
printf("enter a valid infix expression\n");
scanf("%s",infix);
infix_postfix(infix,postfix);
printf("the postfix expression is\n");
printf("%s\n",postfix);

}
```

5. Design, develop, and implement a program in C for the following stack applications

a. Evaluation of suffix expression with single digit operands and operators: +, - , *, /, % , ^

b. Solving Tower of Hanoi problem with n disks.

```
#include<stdio.h>
#include<string.h>

#include<math.h>
int count=0, top=-1;
int operate(char symb, int op1, int op2)
{
        switch(symb)
        {
                case '+':return op1+op2;
                case '-':return op1-op2;
                case '/':return op1/op2;
                case '*':return op1*op2;
                case '%':return op1%op2;
                case '^':return pow(op1,op2);
        }
}
void push(int stack[],int d)
{
        stack[++top]=d;
}
int  pop(int stack[])
{
        return(stack[top--]);
}
void tower( int n,char src, char intr, char des)
{
        if(n)
        {
                tower(n-1,src,des,intr);
                printf("disk %d  moved from %c to %c\n",n,src,des);
                count++;
                tower(n-1,intr,src,des);
        }
}
void main()
{
        int n, choice,i,op1,op2,ans,stack[50];
        char expr[20],symb;
        while(1)
        {
                printf("\nprogram to perform evaluation of suffix expression and tower of hanoi problem\n");
                printf("\n1.evaluate suffix expression\n2.Tower of hanoi\n3.Exit\n ");
                printf("\nenter the choice\n");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1: printf("enter the suffix expression : ");
                                scanf("%s",expr);
                                for(i=0;expr[i]!= '\0';i++)
                                {
                                        symb=expr[i];
                                        if(symb>='0' && symb<='9')
                                        push(stack, symb-'0');
                                        else
                                        {
```

```
                                                op2=pop(stack);
                                                op1=pop(stack);
printf("given expr is %d %d %c\n",op2,op1,symb);

                                                ans=operate(symb,op1,op2);
                                                push(stack,ans);
                                        }
                                }
                                ans=pop(stack);
                                printf("The result of the suffix expression is  %d",ans);
                        break;

                        case 2:   printf("enter the number of disks\n");
                                scanf("%d",&n);
                                tower(n,'a','b','c');
                                printf("number of moves taken to move disks from source to destination
%d",count);
                        break;
                        case 3: return;
                }
        }
}
```

b. Solving Tower of Hanoi problem with n disks.

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
int count=0,top=-1;
int operate(char symb,int r)
{
switch (symb)
{
case '+':return op1+op2;
case '-':return op1-op2;
case '*':return op1*op2;
case '%':return op1%op2;
case '/':return op1/op2;
case '^':return  pow(op1^op2);
}
}
void push(int stack[],int d)
stack[++top]=d;
int pop(int stack[]);
return (stack[top--]);
void tower(int n,char src,char intr,char des)
{
if(n)
{
tower (n-1,src,des,intr);
printf("disk % moved from %c to% \n",n,src,des);
count ++;
tower (n-1,intr,src,des);
}
}
void main()
{
```

```
int n,choice,op1,op2,ans,stack[50];
char expr[20],symb;
clrscr();
while(1)
{
printf("program to evaluate suffix expression\n");
printf("1.enter the  expression\n  2.tower of hanoi 3.exit\n");
printf("Enter the choice\n");
scanf("%d",&choice);
}
switch (choice)
{
case 1:printf("enter the suffix expression\n");
scanf("%s",expression);
for(i=0,expr[i]!='\0';i++)
{
symb=expr[i];
if(symb>='0'&&symb<=9)
push(stack,symb,'0')
else
{
op2=pop(stack);
op1=pop(stack);
printf("%d %d %c\n",op2.op1,symb);
ans=operate (symb,op1,op2)
push(stack,ans);
}
}
ans=pop(stack);
printf("the rewsulatnt string is %d\n");
break;
case 2:printf("enter the number of disk\n");
scanf("%d",&n);
tower(n,'a','b','c');
printf("the number of string is moved from source file to destination file \n");
break;

case 3:exit(0);
}
}
}
```

6. Design, Develop and Implement a menu driven Program in C for the following operations on Circular QUEUE of Characters (Array Implementation of Queue with maximum size MAX)

a. Insert an Element on to Circular QUEUE
b. Delete an Element from Circular QUEUE
c. Demonstrate *Overflow* and *Underflow* situations on Circular QUEUE
d. Display the status of Circular QUEUE
e. Exit

Support the program with appropriate functions for each of the above operations.

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#define MAX 5
int front=0,rear=-1,count=0,it1;
char cqueue[MAX],element;
void insert();
void delete();
void display();

void main()
{
int choice;

while(1)
{
printf("\n\nprogram to ililstrate operations on CIRCULAR QUEUE of characters\n");
printf("\n\t1=>insert an element on to CIRCULAR QUEUE\n\t2=>delete an element from CIRCULAR QUEUE\n\t3=>display the status of CIRCULAR QUEUE\n\t4=>exit\n");
scanf("%d",&choice);
switch(choice)
{
case 1:insert();
break;
case 2:delete();
break;
case 3:display();
break;
case 4:return;
}
}
}

void insert()
{

if(count==MAX)
{
printf("CIRCULAR QUEUE is full,elements can not be inserted\n");
return;
}
rear=(rear+1)%MAX;
printf("\n enter the element to be inserted into the CIRCULAR QUEUE\n");
element=getche();
cqueue[rear]=element;
count++;
}

void delete()
{
```

```
if(count==0)
{
printf("CIRCULAR QHEUE is empty,no element to delete\n");
return;
}
it1=cqueue[front];
front=(front+1)%MAX;
printf("the element deleted is %c\n",it1);
count-=1;
}

void display()
{
int i;
if(count==0)
{
printf("CIRCULAR QUEUE is empty , no element to display\n");
return;
}
printf("CIRCULAR QUEUE contants are\n");
for(i=front;i<=count;i++)
printf("%c",cqueue[i]);
}
```

7. Design, Develop and Implement a menu driven Program in C for the following operations on Singly Linked List (SLL) of Student Data with the fields: *USN, Name, Branch, Sem, PhNo*
a. Create a SLL of N Students Data by using *front insertion*.
b. Display the status of SLL and count the number of nodes in it
c. Perform Insertion and Deletion at End of SLL
d. Perform Insertion and Deletion at Front of SLL
e. Demonstrate how this SLL can be used as STACK and QUEUE
f. Exit

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

typedef struct
{
int usn;
char name[20];
char branch[20];
int semester;
char phone[20];
}STUDENT;

struct node
{
int usn;
char name[20];
char branch[20];
int semester;
char phone[20];
struct node *link;
};

typedef struct node*NODE;

NODE getnode()
{
NODE x;
x=(NODE)malloc(sizeof(struct node));
if(x==NULL)
{
printf("out of memory\n");
exit(0);
}
return x;
}

NODE insert_front(STUDENT item,NODE first)
{
NODE temp;
temp=getnode();
temp->usn=item.usn;
strcpy(temp->name,item.name);
strcpy(temp->branch,item.branch);
temp->semester=item.semester;
strcpy(temp->phone,item.phone);
temp->link=NULL;
if(first==NULL)
return temp;
temp->link=first;
return temp;
```

```
}

NODE insert_rear(STUDENT item,NODE first)
{
NODE temp,cur;
temp=getnode();
temp->usn=item.usn;
strcpy(temp->name,item.name);
strcpy(temp->branch,item.branch);
temp->semester=item.semester;
strcpy(temp->phone,item.phone);
temp->link=NULL;
if(first==NULL)
return temp;
cur=first;
while(cur->link!=NULL)
{
cur=cur->link;
}
cur->link=temp;
return first;
}
NODE delete_front(NODE first)
{
NODE temp;
if(first==NULL)
{
printf("student list is empty\n");
return NULL;
}
temp=first;
temp=temp->link;
printf("delete student record:USN=%d\n",first->usn);
free(first);
return temp;
}

NODE delete_rear(NODE first)
{
NODE cur,prev;
if(first==NULL)
{
printf("student list is empty cannot delete\n");
return first;
}
if(first->link==NULL)
{
printf("delete student record:USN=%d\n",first->usn);
free(first);
return NULL;
}
prev=NULL;
cur=first;
while(cur->link!=NULL)
{
prev=cur;
cur=cur->link;
}
printf("delete student record:USN=%d\n",cur->usn);
free(cur);
```

```
prev->link=NULL;
return first;
}

void display(NODE first)
{
NODE cur;
int count=0;
if(first==NULL)
{
printf("student list is empty\n");
return;
}
cur=first;
while(cur!=NULL)
{
printf("%d\t%s\t%s\t%d\t%s\t\n",cur->usn,cur->name,cur->branch,cur->semester,cur->phone);
cur=cur->link;
count++;
}
printf("numbrt of students=%d\n",count);
}
void main()
{
NODE first;
int choice;
STUDENT item;
first=NULL;
clrscr();
for(;;)
{
printf("1.insert_front\n2.insert_rear\n3.delete_front\n4.delete_rear\n5.display\n6.exit\n");
printf("Enter the choice\n");
scanf("%d",&choice);
switch(choice)
{
case 1:
        printf("USN       :\n");
        scanf("%d",&item.usn);
        printf("name      :\n");
        scanf("%s",item.name);
        printf("branch    :\n");
        scanf("%s",item.branch);
        printf("semester:\n");
        scanf("%d",&item.semester);
        printf("phone     :\n");
        scanf("%s",item.phone);
        first=insert_front(item,first);
        break;
case 2:
        printf("USN       :\n");
        scanf("%d",&item.usn);
        printf("name      :\n");
        scanf("%s",item.name);
        printf("branch    :\n");
        scanf("%s",item.branch);
        printf("semester:\n");
        scanf("%d",&item.semester);
        printf("phone     :\n");
        scanf("%s",item.phone);
```

```
            first=insert_rear(item,first);
            break;
case 3:
            first=delete_front(first);
            break;
case 4:
            first=delete_rear(first);
            break;
case 5:
            display(first);
            break;
default:
            exit(0);
}
}
}
```

8. Design, Develop and Implement a menu driven Program in C for the following operations on Doubly Linked List (DLL) of Employee Data with the fields: *SSN, Name, Dept, Designation, Sal, PhNo*

a. Create a DLL of N Employees Data by using *end insertion*.
b. Display the status of DLL and count the number of nodes in it
c. Perform Insertion and Deletion at End of DLL
d. Perform Insertion and Deletion at Front of DLL
e. Demonstrate how this DLL can be used as Double Ended Queue
f. Exit

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct node
{
  int ssn;
  char name[20],department[20],designation[20],phone[20];
  float salary;
  struct node *llink;
  struct node *rlink;
};
typedef struct node *NODE;
struct node *item;
NODE getnode()
{
  NODE x;
  x=(NODE)malloc(sizeof(struct node));
  if(x==NULL)
  {
    printf("out of memory\n");
    exit(0);
  }
  return x;
}
void read()
{
item=getnode();
                printf("ssn:");
                scanf("%d",&item->ssn);
                printf("name:");
                scanf("%s",item->name);
                printf("department:");
               scanf("%s",item->department);
                printf("designation:");
              scanf("%s",item->designation);
                printf("salary:");
                scanf("%f",&item->salary);
                printf("phone:");
                scanf("%s",item->phone);
item->llink=item->rlink=NULL;
}
NODE insert_front(NODE first)
{
          read();
 if(first==NULL) return item;
 item->rlink=first;
 first->llink=item;
 return item;
}
NODE insert_rear(NODE first)
```

```
{
 NODE cur;
 read();
 if(first==NULL) return item;
 cur=first;
 while(cur->rlink!=NULL)
 {
  cur=cur->rlink;
 }
 cur->rlink=item;
 item->llink=cur;
 return first;

}
NODE delete_front(NODE first)
{
  NODE second;
  if(first==NULL)
  {
    printf("employee list is empty\n");
    return NULL;
  }
  if(first->rlink==NULL)
  {
  printf("employee details deleted:ssn:=%d\n",first->ssn);
  free(first);
  return NULL;
}
second=first->rlink;
second->llink=NULL;
printf("employee details deleted:ssn:=%d\n",first->ssn);
free(first);
return second;
}
NODE delete_rear(NODE first)
{
  NODE cur,prev;
  if(first==NULL)
  {
    printf("list is empty cannot delete\n");
    return first;
  }
  if(first->rlink==NULL)
  {
     printf("employee details deleted:ssn:=%d\n",first->ssn);
     free(first);
     return NULL;
  }
  prev=NULL;
  cur=first;
  while(cur->rlink!=NULL)
  {
    prev=cur;
    cur=cur->rlink;
  }
  printf("employee details deleted:ssn:=%d\n",cur->ssn);
  free(cur);
  prev->rlink=NULL;
  return first;
}
```

```
void display(NODE first)
{
  NODE temp,cur;
  int count=0;
  if(first==NULL)
  {
    printf("employee list is empty\n");
    return;
  }
  cur=first;
  while(cur!=NULL)
  {
    printf("%d  %f   %s  %s  %s %s\n",cur->ssn,cur->salary,cur->name,cur->department,cur->designation,cur-
>phone);
    cur=cur->rlink;
    count++;
  }
  printf("number of employees=%d\n",count);
}
void main()
{
  NODE first;
  int choice;
  first=NULL;
  for(;;)
  {
    printf("1:insert_front\n2:insert_rear\n");
    printf("3:delete_front\n4:delete_rear\n");
    printf("5:display\n6:exit\n");
    printf("enter the choice\n");
    scanf("%d",&choice);
    switch(choice)
    {
          case 1:first=insert_front(first);
                 break;
          case 2:first=insert_rear(first);
                 break;
          case 3:first=delete_front(first);
                 break;
          case 4:first=delete_rear(first);
                 break;
          case 5:display(first);
                 break;
          default:exit(0);
    }
  }
}
```

9. Design, Develop and Implement a Program in C for the following operations on Singly Circular Linked List (SCLL) with header nodes a. Represent and Evaluate a Polynomial P(x,y,z) = 6x2y2z-4yz5+3x3yz+2xy5z-2xyz3 b. Find the sum of two polynomials POLY1(x,y,z) and POLY2(x,y,z) and store the result in POLYSUM(x,y,z) Support the program with appropriate functions for each of the above operations

```
#include<stdio.h>

#include<math.h>

#include<stdlib.h>

#include<math.h>
typedef struct node

{

int expo,coef; struct node *next; }node;

node * insert(node *,int,int); node * create();

node * add(node *p1,node *p2); int eval(node *p1);

void display(node *head);

node *insert(node*head,int expo1,int coef1)

{

node *p,*q;

p=(node *)malloc(sizeof(node)); p->expo=expo1; p->coef=coef1; p->next=NULL; if(head==NULL)

{

head=p; head->next=head; return(head);

}

if(expo1>head->expo)

{

p->next=head->next; head->next=p; head=p; return(head);


}

if(expo1==head->expo)

{

head->coef=head->coef+coef1; return(head);

}

q=head;
 while(q->next!=head&&expo1>=q->next->expo)
q=q->next;

if(p->expo==q->expo)
```

```
q->coef=q->coef+coef1;
else

{

p->next=q->next; q->next=p;

}

return(head);

}

node *create()

{

int n,i,expo1,coef1; node *head=NULL;

printf("\n\nEnter no of terms of polynomial==>");
scanf("%d",&n);

for(i=0;i<n;i++)

{

printf("\n\nEnter coef & expo==>"); scanf("%d%d",&coef1,&expo1); head=insert(head,expo1,coef1);

}

return(head);

}

node *add(node *p1,node *p2)

{

node *p;

node *head=NULL;

printf("\n\n\nAddition of polynomial==>"); p=p1->next;

do

{

head=insert(head,p->expo,p->coef); p=p->next;

}while(p!=p1->next); p=p2->next;

do

{

head=insert(head,p->expo,p->coef); p=p->next;

}while(p!=p2->next); return(head);
```

```
}
int eval(node *head)
{
node *p; int x,ans=0;
printf("\n\nEnter the value of x="); scanf("%d",&x);
p=head->next; do
{
ans=ans+p->coef*pow(x,p->expo); p=p->next; }while(p!=head->next); return(ans);
}
void display(node *head)
{
node *p,*q; int n=0; q=head->next; p=head->next; do
{
n++; q=q->next;
}while(q!=head->next); printf("\n\n\tThe polynomial is==>"); do
{
if(n-1)

{
printf("%dx^(%d) + ",p->coef,p->expo); p=p->next;
}
else
{
printf(" %dx^(%d)",p->coef,p->expo); p=p->next;
}
n--;
} while(p!=head->next);
}
void main()
{
int a,x,ch;
```

```
node *p1,*p2,*p3; p1=p2=p3=NULL; while(1)
{
printf("\n\t----------<< MENU >>----------"); printf("\n\tPolynomial Operations :");
printf(" 1.Add"); printf("\n\t\t\t\t2.Evaluate"); printf("\n\t\t\t\t3.Exit");
printf("\n\t------------------------------------------- "); printf("\n\n\n\tEnter your choice==>"); scanf("%d",&ch);
switch(ch)
{
case 1 : p1=create(); display(p1); p2=create(); display(p2); p3=add(p1,p2); display(p3); break;
case 2 : p1=create(); display(p1);

a=eval(p1);
printf("\n\nValue of polynomial=%d",a); break;
case 3 : exit(0); break; default :
printf("\n\n\tinvalid choice"); break;
}
}
}
```

10. Design, Develop and Implement a menu driven Program in C for the following operations on Binary Search Tree (BST) of Integers
a. Create a BST of N Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2
b. Traverse the BST in Inorder, Preorder and Post Order
c. Search the BST for a given element (KEY) and report the appropriate message
d. Delete an element(ELEM) from BST
e. Exit

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
        int info;
        struct node*llink;
        struct node*rlink;
};
typedef struct node*NODE;
NODE getnode()
{
        NODE x;
        x=(NODE)malloc(sizeof(struct node));
        if(x==NULL)
        {
         printf("out of memory\n");
         exit(0);
        }
        return x;
}
void preorder(NODE root)
{
        if(root==NULL)return;
        printf("%d  ",root->info);
        preorder(root->llink);
        preorder(root->rlink);
}
void postorder(NODE root)
{
        if(root==NULL)
        return;
        postorder(root->llink);
        postorder(root->rlink);
        printf("%d  ",root->info);
}
void inorder(NODE root)
{
        if(root==NULL)
        return;
        inorder(root->llink);
        printf("%d  ",root->info);
        inorder(root->rlink);
}
void display(NODE root,int level)
{
        int i;
        if(root==NULL)
        return;
        display(root->rlink,level+1);
        for(i=0;i<level;i++)
        printf("       ");
        printf("%d\n",root->info);
```

```
        display(root->llink,level+1);
}
NODE insert(int item,NODE root)
{
        NODE temp,cur,prev;
        temp=getnode();
        temp->info=item;
        temp->llink=NULL;
        temp->rlink=NULL;
        if(root==NULL)
        return temp;
        prev=NULL;
        cur=root;
        while(cur!=NULL)
        {
          prev=cur;
          if(item<cur->info)
          cur=cur->llink;
          else
          cur=cur->rlink;
        }
        if(item<prev->info)
        prev->llink=temp;
        else
        prev->rlink=temp;
        return root;
}
NODE search(int item,NODE root)
{
        NODE cur;
        if(root==NULL)
        return NULL;
        cur=root;
        while(cur!=NULL)
        {
          if(item==cur->info)
          return cur;
          if(item<cur->info)
          cur=cur->llink;
          else
          cur=cur->rlink;
        }
        return NULL;
}
void main()
{
        NODE root,cur;
        int choice,item;
        root=NULL;
        clrscr();
        for(;;)
        {
           printf("1.insert 2.preorder\n");
           printf("3.postorder        4.inorder\n");
           printf("5.search           6.exit\n");
           printf("enter the choice\n");
           scanf("%d",&choice);
           switch(choice)
           {
             case 1:printf("enter the item to be inserted\n");
```

```
                scanf("%d",&item);
                root=insert(item,root);
                break;
        case 2:if(root==NULL)
                {
                    printf("tree is empty\n");
                    break;
                }
                printf("the given tree in tree form is\n");
                display(root,1);
                printf("preorder traversing is\n");
                preorder(root);
                printf("\n");
                break;
        case 3:if(root==NULL)
                {
                    printf("tree is empty\n");
                    break;
                }
                printf("the given tree in tree form is\n");
                display(root,1);
                printf("postorder traversing is\n");
                postorder(root);
                printf("\n");
                break;
        case 4:if(root==NULL)
                {
                    printf("tree is empty\n");
                    break;
                }
                printf("the given tree in tree form is\n");
                display(root,1);
                printf("inorder traversing is\n");
                inorder(root);
                printf("\n");
                break;
        case 5:printf("enter the item to be search\n");
                scanf("%d",&item);
                cur=search(item,root);
                if(cur==NULL)
                printf("item not found\n");
                else
                printf("item found\n");
                break;
        default:exit(0);
      }
    }

}
```

11. Design, Develop and Implement a Program in C for the following operations on Graph(G) of Cities
a. Create a Graph of N cities using Adjacency Matrix.
b. Print all the nodes reachable from a given starting node in a digraph using BFS method
c. Check whether a given graph is connected or not using DFS method.

```
#include<stdio.h>
#include<conio.h>
int a[10][10],s[10],n;
void bfs(int u)
{
        int f,r,q[10],v;
        printf("the nodes visited from %d ",u);
        f=0,r=-1;
        q[++r]=u;
        s[u]=1;
        printf("%d ",u);
        while(f<=r)
        {
                u=q[f++];
                for(v=0;v<n;v++)
                {
                        if(a[u][v]==1)
                        {
                                if(s[v]==0)
                                {
                                        printf(" %d ",v);
                                        s[v]=1;
                                        q[++r]=v;
                                }
                        }
                }
        }
        printf("\n");
}
void dfs(int u)
{
        int v;
        s[u]=1;
        printf(" %d ",u);
        for(v=0;v<n;v++)
        {
                if(a[u][v]==1&&s[v]==0)
                dfs(v);
        }
}
void main()
{
                int i,j,choice,source,s1;
                clrscr();
                printf("enter the no of nodes\n");
                scanf("%d",&n);
                printf("enter the adjacency matrix\n");
                for(i=0;i<n;i++)
                {
                        for(j=0;j<n;j++)
                        scanf("%d",&a[i][j]);
                }
                for(;;)
```

```
                {
                    printf("\n1:reachable nodes using bfs\n 2:reachable nodes using dfs\n3:exit\n");
                    scanf("%d",&choice);
                    switch(choice)
                    {

                        case 1:printf("enter the source node\n");
                            scanf("%d",&s1);
                            bfs(s1);
                            break;
                        case 2:for(source=0;source<n;source++)
                            {
                                for(i=0;i<n;i++)
                                s[i]=0;
                                printf("\n reachable node from %d: ",source);
                                dfs(source);
                            }
                        break;
                        case 3:exit(0);
                    }

                }
}
```

12. Given a File of N employee records with a set K of Keys(4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table(HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are Integers. Design and develop a Program in C that uses Hash function H: K ®L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
#define HASH_SIZE 5
typedef struct empolyee
{
        int id;
        char name[20];
        char des[20];
}EMPLOYEE;
void intialize_hash_table(EMPLOYEE a[])
{
        int i;
        for(i=0;i<HASH_SIZE;i++)
        {
                a[i].id=0;
  //            a[i].name={""};
//              a[i].des=NULL;
        }
}
void insert_hash_table(int id,char des[],char name[],EMPLOYEE a[])
{
        int i,index,h_value;
        h_value=id%HASH_SIZE;
        for(i=0;i<HASH_SIZE;i++)
        {
                index=(h_value+i)%HASH_SIZE;
                if(a[index].id==0)
                {
                        a[index].id=id;
                        strcpy(a[index].name,name);
                        strcpy(a[index].des,des);
                        break;
                }
        }
        if(i==HASH_SIZE)
        printf("table full");
}
int search_hash_table(int key,EMPLOYEE a[])
{
        int i,index,h_value;
        h_value=key%HASH_SIZE;
        for(i=0;i<HASH_SIZE;i++)
        {
                index=(h_value+i)%HASH_SIZE;
                if(key==a[index].id)
                return 1;
                if(a[index].id==0)
                return 0;
        }
        if(i==HASH_SIZE)
```

```
	return 0;
}


void display_hash_table(EMPLOYEE a[],int n)
{
	int i;
	printf("\n\tid\tname\tdes\n");
	for(i=0;i<n;i++)
	{
		if(a[i].id!=0)
		printf("a[%d]=%d\t%s\t%s\n",i,a[i].id,a[i].name,a[i].des);
	}
}
void main()
{
	EMPLOYEE a[10];
	char name[20],des[20];
	int key,id,choice,flag;
	clrscr();
	intialize_hash_table(a);
	for(;;)
	{
		printf("1:insert\n2:search\n3:display\n4:exit\n");
		scanf("%d",&choice);
		switch(choice)
		{
			case 1:printf("enter emp id emp name  des\n");
				scanf("%d%s%s",&id,name,des);
				insert_hash_table(id,des,name,a);
				break;
			case 2:printf("enter key");
				scanf("%d",&key);
				flag=search_hash_table(key,a);
				if(flag==0)
				printf("key not found");
				else
				printf("key found");
				break;
			case 3:printf("contents of hash table are");
				display_hash_table(a,HASH_SIZE);
				break;
			default:exit(0);
		}
	}
}
```