ವಿಶ್ವೇಶ್ವರಯ್ಯ ತಾಂತ್ರಿಕ ವಿಶ್ವವಿದ್ಯಾಲಯ, ಬೆಳಗಾವಿ
VISVESVARAYA TECHNOLOGICAL UNIVERSITY - BELAGAVI

# BANGALORE INSTITUTE OF TECHNOLOGY

**K.R.ROAD, V.V.PURA, BENGALURU -560 004**

## Department of Computer Science and Engineering

# BCSL305

# Data Structures Laboratory Manual
# III- Semester


## Prepared By
**Dr. Hemavathi P**
**Prof. Neetha Das**
**Prof. Vidyashree A L**

BANGALORE INSTITUTE OF TECHNOLOGY
(*Affiliated to Visvesvaraya Technological University – VTU)*
K.R. Road, V.V. Puram, Bengaluru – 560 004. Phone: 26613237/26615865 Fax: 22426796
**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

# Vision

Establish and develop the Institute as the Centre of higher learning, ever abreast with expanding horizon of knowledge in the field of Engineering and Technology with entrepreneurial thinking, leadership excellence for life-long success and solve societal problems.

# Mission

1. Provide high quality education in the Engineering disciplines from the undergraduate through doctoral levels with creative academic and professional programs.
2. Develop the Institute as a leader in Science, Engineering, Technology, Management and Research and apply knowledge for the benefit of society.
3. Establish mutual beneficial partnerships with Industry, Alumni, Local, State and Central Governments by Public Service Assistance and Collaborative Research.
4. Inculcate personality development through sports, cultural and extracurricular activities and engage in social, economic and professional challenges.

# BANGALORE INSTITUTE OF TECHNOLOGY

(*Affiliated to Visvesvaraya Technological University – VTU* )

K.R. Road, V.V. Puram, Bengaluru – 560 004. Phone: 26613237/26615865 Fax: 22426796


## DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

### VISION:

To be a center of excellence in computer engineering education, empowering graduates as highly skilled professionals.

### MISSION:

1. To provide a platform for effective learning with emphasis on technical excellence.
2. To train the students to meet current industrial standards and adapt to emerging technologies.
3. To instill the drive for higher learning and research initiatives.
4. To inculcate the qualities of leadership and Entrepreneurship.

### PROGRAM EDUCATIONAL OBJECTIVES (PEO)

1. Graduates will apply fundamental and advanced concepts of Computer Science and Engineering for solving real world problems.
2. Graduates will build successful professional careers in various sectors to facilitate societal needs.
3. Graduates will have the ability to strengthen the level of expertise through higher studies and research.
4. Graduates will adhere to professional ethics and exhibit leadership qualities to become an Entrepreneur.

### PROGRAM SPECIFIC OUTCOMES (PSOs)

1. The graduates of the program will have the ability to build software products by applying theoretical concepts and programming skills.
2. The graduates of the program will have the ability to pursue higher studies and also to build mathematical model in research.

## 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.

# BANGALORE INSTITUTE OF TECHNOLOGY

(*Affiliated to Visvesvaraya Technological University – VTU)*

K.R. Road, V.V. Puram, Bengaluru – 560 004. Phone: 26613237/26615865 Fax: 22426796

## DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING


**Course Overview**

**SUBJECT**: DATA STRUCTURES LABORATORY **SUBJECT CODE**: BCSL305

In computer science, a data structure is a data organization, management, and storage format that enable 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 analyzed 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 Btree 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 LEARNING OBJECTIVES (CLO)

CLO 1. Dynamic memory management

CLO 2. Linear data structures and their applications such as stacks, queues and lists

CLO 3. Non-Linear data structures and their applications such as trees and graphs

## COURSE OUTCOMES (CO)

At the end of the course the student will be able to:

CO1. Analyze various linear and non-linear data structures

CO2. Demonstrate the working nature of different types of data structures and their applications

CO3. Use appropriate searching and sorting algorithms for the give scenario.

CO4. Apply the appropriate data structure for solving real world problems

## CO TO PO & PSO MAPPING

**1/2/3**

| | | PO1 | PO2 | PO3 | PO4 | PO5 | PO6 | PO7 | PO8 | PO9 | PO10 | PO11 | PO12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **BCSL305** | **CO1** | 3 | 2 | 3 | | 2 | | | | | | | 3 |
| | **CO2** | 3 | 2 | 3 | | 2 | | | | | | | |
| | **CO3** | 3 | 2 | 3 | | 2 | | | | | | | |
| | **CO4** | 3 | 2 | 3 | | 2 | | | | | | | |

| | | PSO1 | PSO2 |
|---|---|---|---|
| **BCSL305** | **CO4** | **2** | **2** |

# Data Structures Laboratory

**Subject Code: BCSL305** **CIE Marks: 50**
**Hours/Week: 0:0:2** **Total Hours: 28**

**List of Programs**

| Sl. No. | Name of Experiment |
|---|---|
| 1. | Develop a Program in C for the following:<br>a. Declare a calendar as an array of 7 elements (A dynamically Created array) to represent 7 days of a week. Each Element of the array is a structure having three fields. The first field is the name of the Day (A dynamically allocated String), the second field is the date of the Day (A integer), the third field is the description of the activity for a particular day (A dynamically allocated String).<br>b. Write functions create (), read () and display (); to create the calendar, to read the data from the keyboard and to print weeks activity details report on screen. |
| 2. | Develop a Program in C for the following operations on Strings.<br>a. Read a main String (STR), a Pattern String (PAT) and a Replace String (REP)<br>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)<br>a. Push an Element on to Stack<br>b. Pop an Element from Stack<br>c. Demonstrate Overflow and Underflow situations on Stack<br>d. Display the status of Stack<br>e. Exit<br>Support the program with appropriate functions for each of the above operations |
| 4. | Develop 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)<br>and alphanumeric operands. |
| 5. | Develop a Program in C for the following Stack Applications<br>a. Evaluation of Suffix expression with single digit operands and operators: +, -, *, /, %, ^<br>b. Solving Tower of Hanoi problem with n disks |
| 6. | Develop a menu driven Program in C for the following operations on Circular QUEUE of Characters (Array Implementation of Queue with maximum size MAX)<br>a. Insert an Element on to Circular QUEUE<br>b. Delete an Element from Circular QUEUE |

| | |
|---|---|
| | c. Demonstrate Overflow and Underflow situations on Circular QUEUE<br>d. Display the status of Circular QUEUE<br>e. Exit Support the program with appropriate functions for each of the above operations |
| 7. | Develop a menu driven Program in C for the following operations on Singly Linked List (SLL) of Student Data with the fields: USN, Name, Programme, Sem, PhNo<br>a. Create a SLL of N Students Data by using front insertion.<br>b. Display the status of SLL and count the number of nodes in it<br>c. Perform Insertion / Deletion at End of SLL<br>d. Perform Insertion / Deletion at Front of SLL(Demonstration of stack)<br>e. Exit |
| 8. | Develop 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<br>a. Create a DLL of N Employees Data by using end insertion.<br>b. Display the status of DLL and count the number of nodes in it<br>c. Perform Insertion and Deletion at End of DLL<br>d. Perform Insertion and Deletion at Front of DLL<br>e. Demonstrate how this DLL can be used as Double Ended Queue.<br>f. Exit |
| 9. | Develop a Program in C for the following operationson Singly Circular Linked List (SCLL) with header nodes<br>a. Represent and Evaluate a Polynomial $P(x,y,z) = 6x^2y^2z-4yz^5+3xyz+2xy\ 5\ z-2xyz\ 3$<br>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. | Develop a menu driven Program in C for the following operations on Binary Search Tree (BST) of Integers .<br>a. Create a BST of N Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2<br>b. Traverse the BST in Inorder, Preorder and Post Order<br>c. Search the BST for a given element (KEY) and report the appropriate message<br>d. Exit |
| 11. | Develop a Program in C for the following operations on Graph(G) of Cities<br>a. Create a Graph of N cities using Adjacency Matrix.<br>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. 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<br>probing. |

# Data Structures Laboratory

**Subject Code: BCSL305** **I.A. Marks: 50**
**Hours/Week: 02** **Total Hours: 28**

## SCHEDULE OF EXPERIMENTS

| Sl. No | Name of Experiment | WEEK |
|---|---|---|
| 1 | Sample programs | Week1 |
| 2 | Create the Calendar using Array | Week2 |
| 3 | String Matching | Week3 |
| 4 | Stack Operations | Week4 |
| 5 | Converting Infix Expression to Postfix Expression | Week5 |
| 6 | Evaluation of Suffix expression and Solving Tower of Hanoi problem with n disks | Week6 |
| 7 | Test 1 | Week7 |
| 8 | Circular Queue Operations | Week8 |
| 9 | Student details using Singly Linked List | Week9 |
| 10 | Employee details using Doubly Linked List | Week10 |
| 11 | Represent and evaluate a polynomial using Singly Circular Linked List with header Nodes | Week11 |
| 12 | Binary Search Tree Construction and traversal techniques | Week12 |
| 13 | DFS/BFS traversals for digraphs to find reachable nodes | Week13 |
| 14 | Hash table and collision resolving using linear probing. | Week 14 |
| 15 | Final Test | Week 15 |

**General Lab Guidelines:**

1. Conduct yourself in a responsible manner at all times in the laboratory. Intentional misconduct will lead to exclusion from the lab.
2. Do not wander around, or distract other students, or interfere with the laboratory experiments of other students.
3. Read the handout and procedures before starting the experiments. Follow all written and verbal instructions carefully.
4. 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 the Class teacher.
5. The workplace has to be tidy before, during and after the experiment.
6. Do not eat food, drink beverages or chew gum in the laboratory.
7. Every student should know the location and operating procedures of all Safety equipment including First Aid Kit and Fire extinguisher.

**DO'S**:-

1. An ID card is a must.
2. Keep your belongings in a designated area.
3. Sign the log book when you enter/leave the laboratory.
4. Records have to be submitted every week for evaluation.
5. The program to be executed in the respective lab session has to be written in the lab observation copy beforehand.
6. After the lab session, shut down the computers.
7. Report any problem in system (if any) to the person in-charge

**DON'TS:-**

1. 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.
2. Do not change the system settings and keyboard keys.
3. Do not upload, delete or alter any software/ system files on laboratory computers.
4. No additional material should be carried by the students during regular labs.
5. Do not open any irrelevant websites in labs.
6. Do not use a flash drive on lab computers without the consent of the lab instructor.
7. Students are not allowed to work in the Laboratory alone or without the 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 relationships 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:

i) Primitive data structures
ii) 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, 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

i) Linear data structures
ii) 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:

i) Array
ii) Linked list
iii) Stacks
iv) Queues

A data structure in which insertion and deletion is not possible in a linear fashion is called as nonlinear data structure. i.e., which does not show the relationship of logical adjacency between the elements is called a 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 the 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:

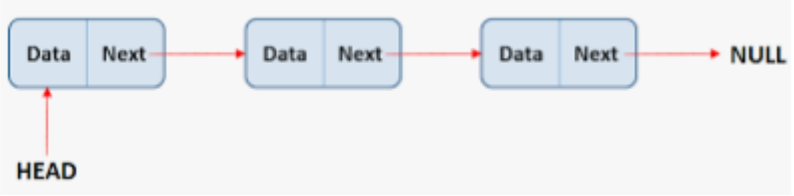
1) Single Linked lists
2) Circular Single Linked Lists
3) Doubly Linked Lists

4) Circular Doubly 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 the 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 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 the 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 has only one link/next, the list is called a 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 a 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 the 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 a list may be contrasted with the static nature of an array, whose size remains constant. When an item has to be inserted, we will have to create a node, which has to be allocated 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 get node 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.

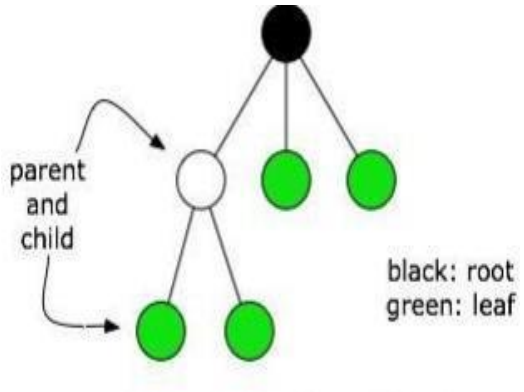



Figure: tree data structure

**Binary tree:**

Tree in which each node has at most two children. 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 nodes of a tree in some order. The different tree traversal techniques are Pre-order, In-order and Post-order traversal. In Preorder 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 an 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 is the tail and v is the head of the edge. Therefore and represent two different edges.

Graph representation can be done using an 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, i1 , i2 …… ik , v such that (u, i1), (i1 , i2) …… (ik , 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.

**SAMPLE PROGRAMS**

1) Program to read and display the array elements

```
#include <stdio.h>
int main()
{
        int values[5];
        int i;
        printf("Enter 5 integers: ");
        for(i = 0; i < 5; ++)                    // taking input and storing it in an array
        {
                scanf("%d", &values[i]);
        }
        printf("Displaying integers:\n");        // printing elements of an array
        for(i = 0; i < 5; ++i)
        {
                printf("%d\n", values[i]);
        }
        getch();
        return 0;
}
```

2) Write a program in C to find the maximum and minimum element in an array.

```
#include   <stdio.h>
#include<conio.h>
void main()
{
        int a[100];
        int i, mx, mn, n;
        printf("Enter the number of elements :");
        scanf("%d",&n);
        printf("Input %d elements in the array: \n",n);
        for(i=0;i<n;i++)
        {
                printf("element - %d: ",i);
                scanf("%d",&a[i]);
        }
        mx = a[0];
        mn = a[0]; for(i=1;
        i<n; i++)
        {
                if(a[i]>mx)
        {
                mx = a[i];
        }
        if(a[i]<mn)
        {
                mn = a[i];
        }
    }
    printf("Maximum element is: %d\n", mx);
    printf("Minimum element is: %d\n\n", mn);
    getch();
}
```

3) Write a program in C to sort elements of array in ascending order using bubble sort.

```
#include    <stdio.h>
#include<conio.h>
void main()
{
        int a[100];
        int n, i, j, temp;
        clrscr();
        printf("\n\nsort elements of array in ascending order using bubble sort:\n ");
        printf("Input the size of array: ");
        scanf("%d", &n);
        printf("Read the elements of array\n”);
        for(i=0;i<n;i++)
        {
                scanf("%d",&a[i]);
         }
         for (i=0 ; i<n-1;i++)
        {
                for(j=0 ;j<n-i-1 ;j++)
                {
                        if(a[j]>a[j+1])
                        {
                                temp=a[j];
                                a[j]=a[j+1];
                                a[j+1]=temp;
                        }
                }
        }
        printf("\nElements of array in sorted ascending order:\n");
        for(i=0; i<n; i++)
        {
                printf("%d \n ", a[i]);
        }
 }
```

4) Write a program in C to concatenate the two strings.

```
#include <stdio.h>
#include<string.h>
void xstrcon(char*,char*);
void main()
{
        int len;
        char s1[]="raj",s2[]="ram";
        xstrcon(s1,s2);
        printf("con = %s\n",s1);
 }
void xstrcon(char *t,char *s)
{
        while(*t)
         t++;
        while(*s)
        {
```

```
                *t++=*s++;
        }
  *t='\0';
}
```

5) Write a program in C to compare the two strings.

```
#include<stdio.h>
#include<string.h>
int xstrcmp(char *,char*);
void main()
{
        char s1[]="raj";
        char s2[]="raj";
        if(xstrcmp(s1,s2)==0)
        printf("same\n");
        else
        printf("not same\n");
}
int xstrcmp(char *s, char *t)
{
        while(*s==*t)
        {
                if(!(*s))
                return 0;
                s++; t++;
        }
                return(*s-*t);
}
```

6) Write a c program to allocate the memory using the dynamic memory Allocation

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void main()
{
        char *str;
        str=(char*)malloc(sizeof(char)*6);
        if(str==NULL)
        {
                printf("out of memory\n");
                exit(1);
        }
        //str=(char*)realloc(str,sizeof(char)*60);
        printf("%u\n",&str);
        strcpy(str,"hi BIT\n");
        printf("%s",str);
        free(str);
        //printf("%s",str);
}
```

7) Write the c program to find the largest element in an array.

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
	int *a,i,j,n;
	printf("enter the num of elements\n");
	scanf("%d",&n);
	a=(int*)malloc(n*sizeof(int));
	printf("enter the elements\n");
	for(i=0;i<n;i++)
	scanf("%d",&a[i]);
	j=0;
	for(i=0;i<n;i++)
	{
		if(a[i]>a[j])
		 j=i;
	}
	printf("the largest element %d found at position %d\n",a[j],j+1);
	free(a);
}
```

## LAB PROGRAMS

### PROGRAM: 1

**Develop a Program in C for the following:**

**a) Declare a calendar as an array of 7 elements (A dynamically Created array) to represent 7 days of a week. Each Element of the array is a structure having three fields. The first field is the name of the Day (A dynamically allocated String), the second field is the date of the Day (A integer), the third field is the description of the activity for a particular day (A dynamically allocated String).**

b) **Write functions create (), read () and display (); to create the calendar, to read the data from the keyboard and to print weeks activity details report on screen**.

```
//Header files
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define NUM_DAYS_IN_WEEK 7

//Global variable Declaration
int i;
// Structure to represent a day
typedef struct
{
        char *acDayName;    // Dynamically allocated string for the day name
        int iDate;          // Date of the day
        char *acActivity; // Dynamically allocated string for the activity description
}
DAYTYPE;

//Prototypes
void  FreeCal(DAYTYPE  *);
void DispCal(DAYTYPE *);
void ReadCal(DAYTYPE *);
DAYTYPE *CreateCal();

//Main function
void main()
{
        // Create the calendar
        DAYTYPE *weeklyCalendar = CreateCal();

        // Read data from the keyboard
        ReadCal(weeklyCalendar);

        // Display the week's activity details
        DispCal(weeklyCalendar);

        // Free allocated memory
        FreeCal(weeklyCalendar);
 }//
```

```
Createcalender function definition DAYTYPE *CreateCal()
{
	DAYTYPE *calendar = (DAYTYPE *)malloc(NUM_DAYS_IN_WEEK * sizeof(DAYTYPE));
	for( i = 0; i < NUM_DAYS_IN_WEEK; i++)
	{
		calendar[i].acDayName = NULL;
		calendar[i].iDate = 0;
		calendar[i].acActivity = NULL;
	}
	return calendar;
}
// Read Calender function definition
void ReadCal(DAYTYPE *calendar)
{
	char Choice;
	for( i = 0; i < NUM_DAYS_IN_WEEK; i++)
	{
		printf("Do you want to enter details for day %d [Y/N]: ", i + 1);
		scanf("%c", &Choice);
		getchar();
		if(tolower(Choice) == 'n')
		continue;
		printf("Day Name: ");
		char nameBuffer[50];
		scanf("%s", nameBuffer);
		calendar[i].acDayName = strdup(nameBuffer); // Dynamically allocate and copy the string
		printf("Date: ");
		scanf("%d", &calendar[i].iDate);
		printf("Activity: ");
		char activityBuffer[100];
		scanf(" %[^\n]", activityBuffer); // Read the entire line, including spaces
		calendar[i].acActivity = strdup(activityBuffer);
		printf("\n");
		getchar(); //remove trailing enter character in input buffer
	}
}
// DisplayCalender function definition
void DispCal(DAYTYPE *calendar)
{
	printf("\nWeek's Activity Details:\n");
	for(i = 0; i < NUM_DAYS_IN_WEEK; i++)
	{
		printf("Day %d:\n", i + 1);
		if(calendar[i].iDate == 0)
		{
			printf("No Activity\n\n");
			continue;
		}
	printf(" Day Name: %s\n", calendar[i].acDayName);
	printf(" Date: %d\n", calendar[i].iDate);
	printf(" Activity: %s\n\n", calendar[i].acActivity);
	}
}
```

```
// FreeCalender function definition
void FreeCal(DAYTYPE *calendar)
{
        for( i = 0; i < NUM_DAYS_IN_WEEK; i++)
        {
                free(calendar[i].acDayName);
                 free(calendar[i].acActivity);
        }
         free(calendar);
}
```

**OUTPUT:**

```
[root@localhost 2022batchDSLab]# cc 1_Calender.c
[root@localhost 2022batchDSLab]# ./a.out

Do you want to enter details for day 1 [Y/N]: y
Day Name: Monday
Date: 27102023
Activity: CSI work

Do you want to enter details for day 2 [Y/N]: y
Day Name: Tuesday
Date: 28102023
Activity: FDP conduction

Do you want to enter details for day 3 [Y/N]: y
Day Name: Wednesday
Date: 29102023
Activity: Proposal Writeup

Do you want to enter details for day 4 [Y/N]: y
Day Name: Thursday
Date: 30102023
Activity: Data Analysis

Do you want to enter details for day 5 [Y/N]: y
Day Name: Friday
Date: 31102023
Activity: Article Review

Do you want to enter details for day 6 [Y/N]: y
Day Name: Saturday
Date: 01112023
Activity: Week Off Enjoy

Do you want to enter details for day 7 [Y/N]: y
Day Name: Sunday
Date: 02112023
Activity: Enjoy


Week's Activity Details:
Day 1:
  Day Name: Monday
  Date: 27102023
  Activity: CSI work

Day 2:
  Day Name: Tuesday
  Date: 28102023
  Activity: FDP conduction

Day 3:
  Day Name: Wednesday
```

Date: 29102023
Activity: Proposal Writeup

Day 4:
Day Name: Thursday
Date: 30102023
Activity: Data Analysis

Day 5:
Day Name: Friday
Date: 31102023
Activity: Article Review

Day 6:
Day Name: Saturday
Date: 1112023
Activity: Week Off Enjoy

Day 7:
Day Name: Sunday
Date: 2112023
Activity: Enjoy

**OUTPUT 2 :**

```
[root@localhost 2022batchDSLab]# ./a.out

Do you want to enter details for day 1 [Y/N]: y
Day Name: Wednesday
Date: 01112023
Activity: Article Review

Do you want to enter details for day 2 [Y/N]: n
Do you want to enter details for day 3 [Y/N]: n
Do you want to enter details for day 4 [Y/N]: n
Do you want to enter details for day 5 [Y/N]: n
Do you want to enter details for day 6 [Y/N]: n
Do you want to enter details for day 7 [Y/N]: y
Day Name: sunday
Date: 07112023
Activity: Enjoy

Week's Activity Details:
Day 1:
Day Name: Wednesday
Date: 1112023
Activity: Article Review

Day 2:
No Activity
Day 3:
No Activity
Day 4:
No Activity
Day 5:
No Activity
Day 6:
No Activity
Day 7:
Day Name: sunday
Date: 7112023
Activity: Enjoy
```

**PROGRAM: 2**

**Develop 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.**

```
// Header files
#include <stdio.h>
#define MAX 100
//Prototype
int StringMatch(char [], char [], char [], char []);

//Main function
int main()
{
        char s[MAX]={0}, p[MAX]={0}, r[MAX]={0}, final[MAX]={0};
        int found;
        printf("Enter Source String : \n");
        gets(s);
        printf("Enter Pattern String : \n");
         gets(p);
        printf("Enter Replace String : \n");
        gets(r);
        found = StringMatch(s, p, r, final);
        if (found == 1)
        {
                printf("The Final String is : \n");
                puts(final);
        }
        else printf("Search string Not Found\n");
        return 0;
}

// StringMatch function definition
int StringMatch(char s[], char p[], char r[], char f[])
{
        int i, j, k, m, t; // i-index src, j-index pattern, k-index replace, t-index final
        int found = 0;
        j = m = i = t = 0;
        while (s[i] != '\0')
        {
                 if (s[m++] == p[j++]) // check for matching
                {
                        if (p[j] == '\0') // pattern found
                        {
                                // copy replace string in final string
                                for(k=0; r[k]!='\0'; k++,t++)
                                f[t] = r[k];
                                j = 0;
                                i = m;
                                found = 1;
```

```
                        }
                    }
                    else // mismatch
                    {
                        f[t++] = s[i++];
                        m = i;
                        j = 0;
                    }
            }
            return found;
}
```

**OUTPUT:**

```
[root@localhost 2022batchDSLab]# cc 2_string.c
[root@localhost 2022batchDSLab]# ./a.out

Enter Source String :
Welcome to bit, cse
Enter Pattern String :
bit
Enter Replace String :
mit
The Final String is :
Welcome to mit, cse

[root@localhost 2022batchDSLab]# ./a.out
Enter Source String :
Bangalore is beautiful but bangalore is crowded
Enter Pattern String :
Bangalore
Enter Replace String :
Mangalore
The Final String is :
Mangalore is beautiful but bangalore is crowded

[root@localhost 2022batchDSLab]# ./a.out
Enter Source String :
Bangalore is called silicon city, Bangalore is also called as garden city
Enter Pattern String :
Bangalore
Enter Replace String :
Mangalore
The Final String is :
Mangalore is called silicon city, Mangalore is also called as garden city

[root@localhost 2022batchDSLab]# ./a.out
Enter Source String :
temple run is boring
Enter Pattern String :
bor
Enter Replace String :
interest
The Final String is :
temple run is interesting

root@localhost 2022batchDSLab]# ./a.out
Enter Source String :
life is easy
Enter Pattern String :
boring
Enter Replace String :
nice
Search string Not Found
```

**PROGRAM: 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 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.**

```
// Header files
#include<stdio.h>
#include<string.h>
#define MAX_SIZE 6
//Prototypes

void push(int[],int*,int);
int pop(int[],int*);
void display(int[],int);
int palindrome(int[],int*,char[]);

//Main function
int main()
{
        int stack[MAX_SIZE],ele,deleted_item,top=-1,flag=-1,done=0,choice;
        char palstr[MAX_SIZE];
        while(!done)
        {
                printf("\nSTACK OPERATIONS USING ARRAY");
                printf("\n1.Push\n2.Pop\n3.Palindrome\n4.Exit\n");
                printf("Enter the choice:\n");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1:printf("Enter the element to be pushed\n");
                                scanf("%d",&ele);
                                push(stack,&top,ele);
                                display(stack,top);
                                break;
                        case 2:deleted_item=pop(stack,&top);
                                if(deleted_item!=-1)
                                {
                                    printf("Deleted Item is: =%d\n",deleted_item);
                                    display(stack,top);
                                  }
                                  break;
                         case 3: printf("Enter the String\n");
```

```
                                        scanf("%s",palstr);
                                        top=-1;
                                        flag=palindrome(stack,&top,palstr);
                                        if(flag==1)
                                        printf("%s is a palindrome\n",palstr);
                                        else
                                        printf("%s is not a palindrom\n",palstr);
                                        top=-1;
                                        break;
                                case 4: done=1;
                                        break;
                                default:printf("Invalid Choice\n");
                        }
                }
        return 0;
}

// Push function definition
void push(int stack[],int *top,int ele)
{
        if(*top==MAX_SIZE-1)
        printf("Stack Overflow\n");
        else
        {
           ++(*top);
           stack[*top]=ele
        }
        return;
}

//Pop function definition
int pop(int stack[],int*top)
{
        int deleted_item;
        if(*top==-1)
        {
           printf("Stack Underflow\n");
            return -1;
        }
        else
        {
           deleted_item=stack[*top];
            (*top)--;
           return deleted_item;
         }
}
//Display function definition
void display(int stack[],int top)
{
        int i;
        if(top==-1)
        printf("Stack is Empty\n");
        else
```

```
        {

                printf("Stack Elements are\n");
                for(i=top;i>=0;i--)
                 printf("%d\t",stack[i]);
        }
        return;
}


//Palindrome function definition
int palindrome(int stack[],int *top,char pal[])
{
        int i,p;
        for(i=0;i<strlen(pal);i++)
        push(stack,top,pal[i]);
        for(i=0;i<strlen(pal);i++)
        {
                if(pal[i]!=pop(stack,top))
                return -1;
        }
        return 1;
}
```

**OUTPUT**

```
[root@localhost 2022batchDSLab]# cc 3_Stack.c
[root@localhost 2022batchDSLab]# ./a.out

STACK OPERATIONS USING ARRAY
1.Push
2.Pop
3.Palindrome
4.Exit
Enter the choice:
1
Enter the element to be pushed
11
Stack Elements are
11

STACK OPERATIONS USING ARRAY
1.Push
2.Pop
3.Palindrome
4.Exit
Enter the choice:
1
Enter the element to be pushed
22
Stack Elements are
22      11

STACK OPERATIONS USING ARRAY
1.Push
2.Pop
3.Palindrome
4.Exit
Enter the choice:
1
Enter the element to be pushed
33
Stack Elements are
33      22      11

STACK OPERATIONS USING ARRAY
1.Push
2.Pop
3.Palindrome
4.Exit
Enter the choice:
1
Enter the element to be pushed
44
```

```
Stack Elements are
44      33      22      11

STACK OPERATIONS USING ARRAY
1.Push
2.Pop
3.Palindrome
4.Exit
Enter the choice:
1
Enter the element to be pushed
55
Stack Elements are
55      44      33      22      11

STACK OPERATIONS USING ARRAY
1.Push
2.Pop
3.Palindrome
4.Exit
Enter the choice:
1
Enter the element to be pushed
66
Stack Elements are
66      55      44      33      22      11

STACK OPERATIONS USING ARRAY
1.Push
2.Pop
3.Palindrome
4.Exit
Enter the choice:
1
Enter the element to be pushed
77
Stack Overflow
Stack Elements are
66      55      44      33      22      11

STACK OPERATIONS USING ARRAY
1.Push
2.Pop
3.Palindrome
4.Exit
Enter the choice:
2
Deleted Item is: =66
Stack Elements are
```

```
55      44      33      22      11

STACK OPERATIONS USING ARRAY
1.Push
2.Pop
3.Palindrome
4.Exit
Enter the choice:
1
Enter the element to be pushed
66
Stack Elements are
66      55      44      33      22      11

STACK OPERATIONS USING ARRAY
1.Push
2.Pop
3.Palindrome
4.Exit
Enter the choice:
1
Enter the element to be pushed
77
Stack Overflow
Stack Elements are
66      55      44      33      22      11

STACK OPERATIONS USING ARRAY
1.Push
2.Pop
3.Palindrome
4.Exit
Enter the choice:
2
Deleted Item is: =66
Stack Elements are
55      44    33   22      11

STACK OPERATIONS USING ARRAY
1.Push
2.Pop
3.Palindrome
4.Exit
Enter the choice:
2
Deleted Item is: =55
Stack Elements are
44      33      22      11
```

STACK OPERATIONS USING ARRAY
1.Push
2.Pop
3.Palindrome
4.Exit
Enter the choice:
2
Deleted Item is: =44
Stack Elements are
33 22 11

STACK OPERATIONS USING ARRAY
1.Push
2.Pop
3.Palindrome
4.Exit
Enter the choice:
2
Deleted Item is: =33
Stack Elements are
22 11

STACK OPERATIONS USING ARRAY
1.Push
2.Pop
3.Palindrome
4.Exit
Enter the choice:
2
Deleted Item is: =22
Stack Elements are
11

STACK OPERATIONS USING ARRAY
1.Push
2.Pop
3.Palindrome
4.Exit
Enter the choice:
2
Deleted Item is: =11
Stack is Empty

STACK OPERATIONS USING ARRAY
1.Push
2.Pop
3.Palindrome
4.Exit

```
Enter the choice:
2
Stack Underflow

STACK OPERATIONS USING ARRAY
1.Push
2.Pop
3.Palindrome
4.Exit
Enter the choice:
3
Enter the String
amma
amma is a palindrome

STACK OPERATIONS USING ARRAY
1.Push
2.Pop
3.Palindrome
4.Exit
Enter the choice:
3
Enter the String
appa
appa is a palindrome

STACK OPERATIONS USING ARRAY
1.Push
2.Pop
3.Palindrome
4.Exit
Enter the choice:

3
Enter the String
tina
tina is not a palindrome
```

**PROGRAM : 4**

**Develop 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.**

```
// Header files
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<string.h>
#define SIZE 100

//Global variables
char stack[SIZE],infix[SIZE];
char symb, x;
int top = -1;

// push function definition
void push(char item)
{
        if(top >= SIZE -1)
        printf("Stack Overflow\n");
        else
        stack[++top] = item;
}

//pop function definition
char pop()
{
         if(top == -1)
         {
                printf("Stack Underflow\n");
                exit(0);
         }
        else
        return stack[top--];
}

//operator precedence function definition
int priority(char symbol)
{
         if(symbol == '(')
        return 0;
        else if(symbol == '+' || symbol == '-')
         return 1;
        else if(symbol == '*' || symbol == '/' || symbol == '%')
         return 2;
        else if(symbol == '^' || symbol == '$')
        return 3;
        else
        return 0;
}

//infix to postfix conversion function definition
void infix_postfix(char infix[])
{
```

```
int i;
printf("Postfix Expression  :");
 for(i=0; infix[i]!='\0'; i++)
{
        symb = infix[i];
        if(isalnum(symb))
        printf("%c ",symb);  /* place the character in postfix expression */
        else if(symb == '(')
        push(symb);
         else if(symb == ')')
        {
                while((x = pop()) != '(')
                printf("%c ", x);
        }
        else
        {
                while(priority(stack[top]) >= priority(symb))
                printf("%c ",pop());
                push(symb);
        }
    }
}

//main function
int main()
{
        printf("ASSUMPTION: The Infix Expression contains single letter variables and single digit     constants
only.\n");
        printf("\nEnter the Infix Expression : ");
        scanf("%s",infix);
        printf("\n");
        push('(');                // push '(' onto stack
        strcat(infix,")");        // append ')' to infix expression where ")" acts as sentinel
        //printf("%s\n", infix);
        infix_postfix(infix);
        return 0;
}
```

**OUTPUT:**

```
[root@localhost 2022batchDSLab]# cc 4_ Infix.c
[root@localhost 2022batchDSLab]# ./a.out

ASSUMPTION: The Infix Expression contains single letter variables and single digit constants only.
Enter the Infix Expression: A^B*c-D+E/F/G/+H
Postfix Expression: A B ^ c * D - E F / G / / + H +


ASSUMPTION: The Infix Expression contains single letter variables and single digit constants only.
Enter the Infix Expression: A+(B*C-(D/E^F)*G)*H
Postfix Expression: A B C * D E F ^ / G * - H * +

ASSUMPTION: The Infix Expression contains single letter variables and single digit constants only.
Enter the Infix expression: (3^2*5)/(3*2-3)+5
Postfix Expression: 3 2 ^ 5 * 3 2 * 3 - / 5 +

ASSUMPTION: The Infix Expression contains single letter variables and single digit constants only.
Enter the Infix Expression: ((A+B)*C-(D-E))^(F+G)
Postfix Expression: A B + C * D E - - F G + ^


ASSUMPTION: The Infix Expression contains single letter variables and single digit constants only.
Enter the Infix Expression: ((A+B)*C-(D-E))$(F+G)
Postfix Expression: A B + C * D E - - F G + $
```

**PROGRAM: 5A**

**Design, Develop and Implement a Program in C for the following Stack Applications**
**a. Evaluation of Suffix expression with single digit operands and operators: +, -, *, /, %, ^**

```
//Header functions
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<ctype.h>

//Global Variables
int i, top = -1;
int op1, op2, res, s[20];
char postfix[90], symb;

// push function defintion
void push (int item)
{
        top = top+1;
        s[top] = item;
}

//pop function defintion
int pop()
{
        int item;
        item = s[top];
        top = top-1;
        return item;
}


//main function
void main()
{
         printf("\nEnter a valid Postfix Expression:\n");
         scanf("%s", postfix);
         for(i=0; postfix[i]!='\0'; i++)
         {
                  symb = postfix[i];
                  if(isdigit(symb))
                  {
                           push(symb - '0'); // Ascii value of zero(0) is 48

                  }// push(symb - 48) also works
                                                          {
                 else
                 {op2 = pop(); op1 = pop(); switch(symb)
                                  case '+': push(op1 + op2);
                                            break;
                                  case '-': push(op1 - op2);

                                  break;
                                  case '*':push(op1 * op2);
```

```
                        break;
                        case '/': if(op2 == 0)
                        {
                                                printf("D
                                                ivide by
                        }                       zero
                        else                    error\n")
                        {                       ; exit(0);
                                                push(op
                                                1 / op2);
                        }                       break;

                        case '%':push(op1 % op2);
                                 break;
                        case '^': push(pow(op1, op2));
                                  break;
                       default : printf("Invalid Expression");
                    }
                }
        }
        res = pop();
        printf("\n Result = %d", res);
}
```

**OUTPUT:**

```
[root@localhost 2022batchDSLab]# cc 5_stack.c -lm
[root@localhost 2022batchDSLab]# ./a.out

 Enter a valid Postfix Expression:
42^3*3-84/11+/+
 Result = 46

 Enter a valid Postfix Expression:
623+-382/+*2^3+
Result = 52

Enter a valid Postfix Expression:
25+22-/
Divide by zero error
```

# Program: 5b

**Design, Develop and Implement a Program in C for the following Stack Applications**

**b. Solving Tower of Hanoi problem with n disks**

```
//Header files
#include<stdio.h>
//Global variable declaration
int count = 0;
//Tower of Hanoi function definition
void TOH(int n,char A,char B,char C)
{
        if(n>0)
        {
                TOH(n-1, A, C, B);
                printf("Disk %d moved from %c --> %c:\n", n, A, C);
                count++;
                TOH(n-1,B,A,C);
                }
}

//main function
  void main()
   {
        int n;
        printf("Enter the number of disks:");
        scanf("%d",&n);
        TOH(n,'A','B','C');
        printf("Number of moves taken to move disks from source to destination:  %d", count);

   }
```

**OUTPUT:**

```
[root@localhost 2022batchDSLab]# cc 5b_Tower of Hanoi.c
[root@localhost 2022batchDSLab]# ./a.out


Enter the number of disks:1
Disk 1 moved from A --> C:
Number of moves taken to move disks from source to destination: 1

Enter the number of disks:2
Disk 1 moved from A --> B:
Disk 2 moved from A --> C:
Disk 1 moved from B --> C:
Number of moves taken to move disks from source to destination: 3


Enter the number of disks:3
Disk 1 moved from A --> C:
Disk 2 moved from A --> B:
Disk 1 moved from C --> B:
Disk 3 moved from A --> C:
Disk 1 moved from B --> A:
Disk 2 moved from B --> C:
Disk 1 moved from A --> C:
Number of moves taken to move disks from source to destination: 7

Enter the number of disks:4
Disk 1 moved from A --> B:
Disk 2 moved from A --> C:
Disk 1 moved from B --> C:
Disk 3 moved from A --> B:
Disk 1 moved from C --> A:
Disk 2 moved from C --> B:
Disk 1 moved from A --> B:
Disk 4 moved from A --> C:
Disk 1 moved from B --> C:
Disk 2 moved from B --> A:
Disk 1 moved from C --> A:
Disk 3 moved from B --> C:
Disk 1 moved from A --> B:
Disk 2 moved from A --> C:
Disk 1 moved from B --> C:
Number of moves taken to move disks from source to destination: 15
```

**PROGRAM: 6**

**Develop 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<stdlib.h>
#define MAX 5
int front = -1, rear = -1; // global variable
char CQueue[MAX];
void Enqueue();
void Dequeue();
void Display();

int main()
{
    int choice;
    while(1)
    {
            printf("\nCIRCULAR QUEUE OPERATIONS USING ARRAY\n\n");
            printf("1.INSERT\n");
            printf("2.DELETE\n");
            printf("3.DISPLAY\n");
            printf("4.EXIT \n");
            printf("Enter your choice : ");
            scanf("%d", &choice);
            switch (choice)
            {
                case 1:Enqueue();
                        break;
                case 2:Dequeue();
                        break;
                case 3:Display();
                        break;
                case 4:exit(1);
                default:printf("\nInvalid Choice\n");
             }
     }
}
void Enqueue()
{
    char item;
    if ((front == 0 && rear == MAX - 1) || front == rear + 1)
    {
            // front==rear + 1 occurs when an element is inserted at
            //  0th position due to circular increment and rear value is just one less than front i.
            //That is front = 1 and rear =0
            printf("\nCircular Queue is Full \n");
    }
```

```
    else
    {
            printf("Enter the element to insert into the Circular Queue :\n ");
            scanf(" %c", &item);
            if (front == -1)
            front = 0;
            rear = (rear + 1) % MAX;
            CQueue[rear] = item;
    }
}

void Dequeue()
{
    if((front==-1) && (rear==-1))  // condition to check queue is empty
    {
            printf("\nCircular Queue is Underflow\n");
    }
    else if(front==rear)  //Q has only one element, so we reset the
             // queue after dequeue operation
    {
            printf("\nThe Dequeued Element is %c:", CQueue[front]);
            front=-1;
             rear=-1;
    }
    else
    {
            printf("\nThe Dequeued Element is %c:", CQueue[front]);
            front=(front+1)%MAX;
    }
}
void Display()
{
    int i;
    if (front == -1 && rear == -1)
            {
               printf("\nThe Circular Queue is Empty \n");
            }
    else
    {
            printf("\nElements in a Circular Queue are :");
           if (front > rear)
            {
                    for (i = front; i < MAX; i++)
                    {
                            printf("%c ", CQueue[i]);
                    }
                    for (i = 0; i <= rear; i++)
                    printf("%c ", CQueue[i]);
            }
      else
     {
          for (i = front; i <= rear; i++)
             printf("%c ", CQueue[i]);
       }
   }
}
```

**OUTPUT :**

```
[root@localhost 2022batchDSLab]# cc 6_Circular QUEUE
[root@localhost 2022batchDSLab]# ./a.out


CIRCULAR QUEUE OPERATIONS USING ARRAY
1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
Enter your choice : 2

Circular Queue is Underflow

CIRCULAR QUEUE OPERATIONS USING ARRAY
1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
Enter your choice : 3

The Circular Queue is Empty

CIRCULAR QUEUE OPERATIONS USING ARRAY
1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
Enter your choice : 1

Enter the element to insert into the Circular Queue :
 H

CIRCULAR QUEUE OPERATIONS USING ARRAY
1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
Enter your choice : 1
Enter the element to insert into the Circular Queue :
 E

CIRCULAR QUEUE OPERATIONS USING ARRAY
1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
Enter your choice : 1

Enter the element to insert into the Circular Queue :
M

CIRCULAR QUEUE OPERATIONS USING ARRAY
1.INSERT
2.DELETE
```

```
3.DISPLAY
4.EXIT
Enter your choice : 3

Elements in a Circular Queue are :H E M

CIRCULAR QUEUE OPERATIONS USING ARRAY
1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
Enter your choice : 1

Enter the element to insert into the Circular Queue :
A

CIRCULAR QUEUE OPERATIONS USING ARRAY
1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
Enter your choice : 1

Enter the element to insert into the Circular Queue :
P

CIRCULAR QUEUE OPERATIONS USING ARRAY
1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
Enter your choice : 1

Circular Queue is Full

CIRCULAR QUEUE OPERATIONS USING ARRAY
1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
Enter your choice : 3

Elements in a Circular Queue are :H E M A P

CIRCULAR QUEUE OPERATIONS USING ARRAY
1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
Enter your choice : 2

The Dequeued Element is:H

CIRCULAR QUEUE OPERATIONS USING ARRAY
1.INSERT
2.DELETE
```

```
3.DISPLAY
4.EXIT
Enter your choice : 2

The Dequeued Element is :E

CIRCULAR QUEUE OPERATIONS USING ARRAY
1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
Enter your choice : 3

Elements in a Circular Queue are :M A P

CIRCULAR QUEUE OPERATIONS USING ARRAY
1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
Enter your choice : 1

Enter the element to insert into the Circular Queue :
 G

CIRCULAR QUEUE OPERATIONS USING ARRAY
1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
Enter your choice : 3

Elements in a Circular Queue are :M A P G

CIRCULAR QUEUE OPERATIONS USING ARRAY
1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
Enter your choice : 1

Enter the element to insert into the Circular Queue :
 O

CIRCULAR QUEUE OPERATIONS USING ARRAY
1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
Enter your choice : 3

Elements in a Circular Queue are :M A P G O

CIRCULAR QUEUE OPERATIONS USING ARRAY
1.INSERT
2.DELETE
```

```
3.DISPLAY
4.EXIT
Enter your choice : 1

Circular Queue is Full

CIRCULAR QUEUE OPERATIONS USING ARRAY
1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
Enter your choice : 3

Elements in a Circular Queue are :M A P G O

CIRCULAR QUEUE OPERATIONS USING ARRAY
1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
Enter your choice : 4
```

**PROGRAM: 7**

**Develop a menu driven Program in C for the following operations on Singly Linked List(SLL) of Student Data with the fields: USN, Name, Programme, 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**

```
//Header files
 #include <stdio.h>
#include <malloc.h>
#include <string.h>
#include <stdlib.h>
#define MAX 30

//malloc() function definition using macros
#define MALLOC(p,s,t) \
p = (t) malloc(s); \ if (p == NULL ) \
{ \
        fprintf(stderr, "Insufficient Memory\n"); \
         exit(EXIT_FAILURE); \
}

//USN, Name, Programme, Sem, PhNo
struct List
{
        int USN;
        char Name[MAX];
        char Programme[MAX]; int Sem;
        char PhNo[MAX]; struct List *link;
 };
 typedef struct List *NODE;

 //Prototypes
 NODE Create (NODE, int);
 NODE InsFront (NODE);
 NODE InsRear (NODE);
 NODE DelFront (NODE);
 NODE DelRear (NODE);
 void Display (NODE);
  void ReadData();

 //Global Variable Declaration
 int usn;                /* student usn */
  char sname[MAX];       /* student name */
  char programme[MAX];          /* Programme */
   int sem;                      /* semester      */
  char phone[MAX];         /* phone no. */

 //main function
 void main()
 {
```

```
        int choice, done;
        NODE fi;          /* FIRST node */
         fi= NULL;
        done = 0;
        int n;    /* no. of students */
        while (!done)
        {
                printf("\n1.Create\t2.InsFront\t3.InsRear\t4.Delete Front\n5.Delete Rear\t6.Display\t7.Exit\n");
                printf("Enter Choice: ");
                scanf("%d",&choice); switch (choice)
                {

                        case 1:  printf("Enter No. of Students: ");
                                 scanf("%d", &n);
                                  fi = Create(fi, n);
                                        break;
                        case 2:  ReadData();
                                 fi = InsFront(fi);
                                      break;
                        case 3:  ReadData();
                                 fi= InsRear(fi);
                                 break;
                        case 4:  fi = DelFront(fi);
                                 break;
                        case 5:  fi = DelRear(fi);
                                 break;
                        case 6:  Display(fi);
                                 break;
                        case 7:  done=1;
                                  break;
                        default: printf("Invalid Choice\n\n");
                                 break;
                }
        }
}

// Read student data function definition
void ReadData()
{
        printf("Enter Student USN: ");
        scanf("%d", &usn);
        printf("Enter Student Name: ");
        scanf("%s",sname);
        printf("Enter Student Programme: ");
        scanf("%s",programme);
        printf("Enter Semester: ");
        scanf("%d", &sem);
        printf("Enter Student Phone: ");
         scanf("%s",phone);
}


// Create a list of student data using front insertion definition
NODE Create(NODE first, int n)
{
```

```
        int i;
        if (first == NULL)
        {
                for (i = 1; i <= n; i++)
                {
                        printf("Enter Student Data <%d>:\n", i); ReadData();
                        first = InsFront(first);
                }
                return first;
        }
        else
        printf("List already has some student data\n");
}

// Insertion of student data from front end
NODE InsFront (NODE first)
{
        NODE q;
        MALLOC(q, sizeof(struct List), NODE);
        q->USN = usn;
        strcpy (q->Name, sname);
        strcpy(q->Programme, programme); q->Sem = sem;
        strcpy(q->PhNo, phone);
        q->link = first;
        first = q; return first;
}

// Insertion of student data from rear end
 NODE InsRear (NODE first)
{
        NODE q, t = first;
        MALLOC(q, sizeof(struct List), NODE);
         q->USN = usn;
        strcpy (q->Name, sname);
        strcpy(q->Programme, programme);
        q->Sem = sem;
        strcpy(q->PhNo, phone);
        q->link = NULL;
        if (!first) first = q;
        else
        {
                while (t->link)
                 t = t->link;
                t->link = q;       /* goto last node */
        }
        return first;
}


// Deletion of student data from front end
NODE DelFront (NODE first)
{

        NODE te;
        if(first == NULL)
```

```
            printf("Can't delete, List is Empty\n");
             else
            {
                        te = first;
                        first= first->link; free(te);
            }
            return first;
}

// Deletion of student data from rear end
NODE DelRear (NODE first)
{
            NODE cur, pred;
            if(first == NULL)
            {
                        printf("Can't delete, List is Empty\n"); return first;
            }
            if(first->link == NULL)
            return first=NULL;

            // more than one node */
            cur = first;
          while (cur->link != NULL)
          {
                        pred = cur;
                        cur = cur->link;
          }
          free(cur);
          pred->link = NULL;
          return first;
 }

/* Displays the contents of the list */
void Display (NODE first)
{
          int count =0;
           while (first)
          {
                        printf("%d\t%s\t%s\t%d\t%s\n", first->USN, first->Name, first->Programme, first->Sem, first-
  >PhNo);
                        first = first->link; count++;
          }
          printf("Count of Nodes = %d\n", count);
}
```

**OUTPUT:**

```
[root@localhost 2022batchDSLab]# cc 7_singlelinkedlist.c
 [root@localhost 2022batchDSLab]# ./a.out

1.Create      2.InsFront      3.InsRear          4.Delete Front
5.Delete Rear    6.Display
                 7.Exit

Enter Choice: 6
Count of Nodes = 0

1.Create      2.InsFront      3.InsRear          4.Delete Front
5.Delete Rear      6.Display          7.Exit

Enter Choice: 4
Can't delete, List is Empty

1.Create      2.InsFront      3.InsRear          4.Delete Front
5.Delete Rear      6.Display          7.Exit

Enter Choice: 5
Can't delete, List is Empty


1.Create      2.InsFront      3.InsRear          4.Delete Front
5.Delete Rear      6.Display          7.Exit

Enter Choice: 1
Enter No. of Students: 2

Enter Student Data<1>:
Enter Student USN: 11
Enter Student Name: Lekhi
Enter Student Programme: cs Enter
Semester: 3
Enter Student Phone: 4567

Enter Student Data <2>:
Enter Student USN: 22
Enter Student Name: Ruchi
Enter Student Programme: Photography
 Enter Semester: 4
Enter Student Phone: 23456


1.Create      2.InsFront      3.InsRear          4.Delete Front
5.Delete Rear      6.Display          7.Exit

Enter Choice: 6
22        Ruchi   Photography     4          23456
11        Lekhi    cs                  3          4567

Count of nodes = 2
```

1.Create 2.InsFront 3.InsRear 4.Delete Front
5.Delete Rear 6.Display 7.Exit

Enter Choice: 2
Enter Student USN: 11
Enter Student Name: Neetha
Enter Student Programme: CS
Enter Semester: 3
Enter Student Phone: 67890123

1.Create 2.InsFront 3.InsRear 4.Delete Front
5.Delete Rear 6.Display 7.Exit

Enter Choice: 6
11 Neetha CS 3 67890123
22 Ruchi Photography 4 23456
11 Lekhi cs 3 4567

Count of nodes = 3

1.Create 2.InsFront 3.InsRear 4.Delete Front
5.Delete Rear 6.Display 7.Exit

Enter Choice: 3
Enter Student USN: 22
Enter Student Name: Vidya
Enter Student Programme: EC
Enter Semester: 3
Enter Student Phone: 556788223

1.Create 2.InsFront 3.InsRear 4.Delete Front
1.5.Delete Rear 6.Display 7.Exit

Enter Choice: 6
11 Neetha CS 3 67890123
22 Ruchi Photography 4 23456
11 Lekhi cs 3 4567
22 Vidya EC 3 556788223

Count of nodes = 4

1.Create 2.InsFront 3.InsRear 4.Delete Front
5.Delete Rear 6.Display

7.Exit

Enter Choice: 4

1.Create 2.InsFront 3.InsRear 4.Delete Front
5.Delete Rear 6.Display 7.Exit

Enter Choice: 6

```
22      Ruchi   Photography   4       23456
11      Lekhi   cs            3       4567
22      Vidya   EC            3       556788223

Count of nodes = 3

1.Create    2.InsFront    3.InsRear    4.Delete Front
5.Delete Rear    6.Display    7.Exit

Enter Choice: 5

1.Create    2.InsFront    3.InsRear    4.Delete Front
5.Delete Rear    6.Display    7.Exit

Enter Choice: 6
22      Ruchi   Photography   4       23456
11      Lekhi   cs            3       4567

Count of nodes = 2

1.Create    2.InsFront    3.InsRear    4.Delete Front
5.Delete Rear    6.Display    7.Exit


Enter Choice: 5

1.Create    2.InsFront    3.InsRear    4.Delete Front
5.Delete Rear    6.Display    7.Exit

Enter Choice: 5

1.Create    2.InsFront    3.InsRear    4.Delete Front
5.Delete Rear    6.Display    7.Exit

Enter Choice: 5

Can't delete, List is Empty

1.Create    2.InsFront    3.InsRear    4.Delete Front
5.Delete Rear    6.Display    7.Exit
Enter Choice: 7
```

**PROGRAM: 8**

**Develop 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**

```
// Header files
#include <stdio.h>
#include <malloc.h>
 #include <string.h>
#include <stdlib.h>
#include<ctype.h>
#define MAX 30

// SSN, Name, Dept, Designation, Sal, PhNo
 struct List
{
        int SSN;
        char Name[MAX];
         char Dept[MAX];
        char Designation[MAX];
         float Sal;
        char PhNo[MAX];
        struct List *prev;
        struct List *next;
};
#define MALLOC(p,s,t)\
{
                p=(t)malloc();\
                if(p==NULL){ \
                printf("Insufficient Memory\n"); \
                exit; \
}
typedef struct List *NODE;

//prototypes
NODE Create   (NODE, int);
NODE InsFront (NODE);
NODE InsRear  (NODE);
NODE DelFront (NODE);
NODE DelRear  (NODE);
void Display  (NODE);
void ReadData();

//Global variable declaration
 int  ssn;                      /* employee id  */
char name[MAX];                 /* employee name */
char dept[MAX];                 /* dept */
char desig[MAX];        /* designation    */
```

```
float salary;               /* salary */
char phone[MAX];            /* phone no. */

// main function
int main()
{
        int choice, done;
        NODE fi;
        int n;               /* no. of employees */
        fi= NULL;
        done = 0;
        while (!done)
        {
                printf("\n1.Create\t\2.InsFront\t3.InsRear\t4.Delete Front\n5.DeleteRear\t6.Display\t7.Exit\n");
                printf("Enter Choice: ");
                scanf("%d", &choice);
                switch (choice)
                {
                        case 1:  printf("Enter No. of Employees: ");
                                 scanf("%d", &n);
                                 fi = Create(fi, n);
                                 break;
                        case 2:  ReadData();
                                 fi = InsFront(fi);
                                 break;
                        case 3:  ReadData();
                                 fi = InsRear(fi);
                                 break;
                        case 4:  fi = DelFront(fi);
                                 break;
                        case 5:  fi = DelRear(fi);
                                 break;
                        case 6:  Display(fi);
                                 break;
                        case 7:  done=1;
                                 break;
                        default: printf("Invalid Choice\n\n");
                                 break;
                }
        }
        return 0;
}

//Read employee details function definition
void ReadData()
{
        printf("Enter Employee SSN: ");
        scanf("%d", &ssn);
        printf("Enter Employee Name: ");
        scanf("%s",name);
        printf("Enter Employee Department: ");
        scanf("%s",dept);
        printf("Enter Designation: ");
        scanf("%s",desig);
        printf("Enter Salary: ");
        scanf("%f", &salary);
        printf("Enter Phone: ");
```

```
        scanf("%s",phone);
 }

NODE Create(NODE first, int n)
{
        int i;    NODE q;
        if (first == NULL)
        {
                for (i = 1; i <= n; i++)
                {
                        printf("Enter Employee Data <%d>:\n", i);
                        ReadData();
                        first = InsFront(first);
                }
                return first;
        }
        else
        printf("List already has some Employee data\n");
}

//Insert employee details from front end function
NODE InsFront(NODE first)
{
        NODE q;
        MALLOC(q, sizeof(struct List), NODE);
        q->SSN = ssn;
        strcpy (q->Name, name);
        strcpy(q->Dept, dept);
        strcpy(q->Designation, desig);
        q->Sal = salary;
        strcpy(q->PhNo, phone);
        q->prev = NULL;
        q->next = first;
        if(first != NULL)
        first->prev = q;
        return q;
}

//Insert employee details from rear end function
NODE InsRear(NODE first)
{
        NODE q, cur;
        cur = first;
        while(cur->next != NULL)
        cur = cur->next;
        MALLOC(q, sizeof(struct List), NODE);
        q->SSN = ssn;
        strcpy (q->Name, name);
        strcpy(q->Dept, dept);
        strcpy(q->Designation, desig);
        q->Sal = salary;
        strcpy(q->PhNo, phone);
        q->next = NULL;
        cur->next = q;
        q->prev = cur;
        return first;
}
```

```
//Delete employee details from front end function

NODE DelFront(NODE first)
{
        NODE cur;
        if(first == NULL)
        {
                printf("Can't delete, List is Empty\n");
                return NULL;
        }
        else
        {
                cur=first;
                first = first->next;
                first->prev = NULL;
                free(cur);
                return first;
        }
}

//Delete employee details from rear end function
NODE DelRear(NODE first)
{
        NODE cur = first;
        if(first == NULL)
        {
                printf("Can't delete, List is Empty\n");
                return NULL;
        }
        if(first->next == NULL)
        return first=NULL;
        else
        {
                while(cur->next != NULL)
                cur = cur->next;
                cur->prev->next = NULL;}
                free(cur);
                return first;
        }

//Display employee details
void Display(NODE first)
{
        int count=0;
        if(first == NULL)
        printf("\nList is Empty\n");
        else
        while (first)
        {
                printf("%d\t%s\t%s\t%s\t%f\t%s\n", first->SSN, first->Name, first->Dept, first->Designation,
first->Sal,first->PhNo);
                first = first->next;
                count++;
        }
        printf("Count of Nodes=%d\n",count);
}
```

**OUTPUT:**

```
[root@localhost 2022batchDSLab]# cc 8_ DoublyLinkedList.c
 [root@localhost 2022batchDSLab]# ./a.out

 1.Create      2.InsFront    3.InsRear     4.Delete Front
 5.Delete Rear 6.Display     7.Exit

 Enter Choice: 4

 Can't delete, List is Empty

 1.Create      2.InsFront    3.InsRear     4.Delete Front
 5.Delete Rear 6.Display     7.Exit

 Enter Choice: 5

 Can't delete, List is Empty

 1.Create      2.InsFront    3.InsRear     4.Delete Front
 5.Delete Rear 6.Display     7.Exit

 Enter Choice: 6

 List is Empty

 Count of Nodes=0

 1.Create      2.InsFront    3.InsRear     4.Delete Front
 5.Delete Rear 6.Display     7.Exit

 Enter Choice: 1

 Enter No. of Employees: 2

 Enter Employee Data <1>:

 Enter Employee SSN: 11
 Enter Employee Name: Hema
 Enter Employee Department: CS
 Enter Designation: AssociateProfessor
 Enter Salary: 3000000
 Enter Phone: 345566

 Enter Employee Data <2>:

 Enter Employee SSN: 22
 Enter Employee Name: Neetha
 Enter Employee Department: EC
 Enter Designation: Associateprofessor
 Enter Salary: 5000000
 Enter Phone: 23455566
```

```
1.Create        2.InsFront      3.InsRear       4.Delete Front
5.Delete Rear   6.Display       7.Exit

Enter Choice: 6

22      Neetha  EC      Associateprofessor      5000000.000000      23455566
11      Hema    CS      AssociateProfessor      3000000.000000      345566

Count of Nodes=2

1.Create        2.InsFront      3.InsRear       4.Delete Front
5.Delete Rear   6.Display       7.Exit

Enter Choice: 2

Enter Employee SSN: 33
Enter Employee Name: Vidya
Enter Employee Department: ME
Enter Designation: Professor
Enter Salary: 6000000
Enter Phone: 123456

1.Create        2.InsFront      3.InsRear       4.Delete Front
5.Delete Rear   6.Display       7.Exit

Enter Choice: 6

33      Vidya   ME      Professor       6000000.000000      123456
22      Neetha  EC      Associateprofessor      5000000.000000      23455566
11      Hema    CS      AssociateProfessor      3000000.000000      345566

Count of Nodes=3

1.Create        2.InsFront      3.InsRear       4.Delete Front
5.Delete Rear   6.Display       7.Exit

Enter Choice: 3

Enter Employee SSN: 44
Enter Employee Name: Tina
Enter Employee Department: EEE
Enter Designation: Assistantprofessor
Enter Salary: 1234489
Enter Phone: 456789

1.Create        2.InsFront      3.InsRear       4.Delete Front
5.Delete Rear   6.Display       7.Exit
```

Enter Choice: 6

| | | | | | |
|---|---|---|---|---|---|
| 33 | Vidya | ME | Professor | 6000000.000000 | 123456 |
| 22 | Neetha | EC | Associateprofessor | 5000000.000000 | 23455566 |
| 11 | Hema | CS | AssociateProfessor | 3000000.000000 | 345566 |
| 44 | Tina | EEE | Assistantprofessor | 1234489.000000 | 456789 |

Count of Nodes=4

1.Create 2.InsFront 3.InsRear 4.Delete Front
5.Delete Rear 6.Display 7.Exit

Enter Choice: 4

1.Create 2.InsFront 3.InsRear 4.Delete Front
5.Delete Rear 6.Display 7.Exit

Enter Choice: 6

| | | | | | |
|---|---|---|---|---|---|
| 22 | Neetha | EC | Associateprofessor | 5000000.000000 | 23455566 |
| 11 | Hema | CS | AssociateProfessor | 3000000.000000 | 345566 |
| 44 | Tina | EEE | Assistantprofessor | 1234489.000000 | 456789 |

Count of Nodes=3

1.Create 2.InsFront 3.InsRear 4.Delete Front
5.Delete Rear 6.Display 7.Exit

Enter Choice: 5

1.Create 2.InsFront 3.InsRear 4.Delete Front
5.Delete Rear 6.Display 7.Exit

Enter Choice: 6

| | | | | | |
|---|---|---|---|---|---|
| 22 | Neetha | EC | Associateprofessor | 5000000.000000 | 23455566 |
| 11 | Hema | CS | AssociateProfessor | 3000000.000000 | 345566 |

Count of Nodes=2

1.Create 2.InsFront 3.InsRear 4.Delete Front
5.Delete Rear 6.Display 7.Exit

Enter Choice: 4

**PROGRAM : 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 2 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.**

```
//Header files
#include<stdlib.h>
#include<stdio.h>
#include<malloc.h>
#include<math.h>
#include<ctype.h>

struct polylist
{
        int coef;
         int x,y,z;
         float sal;
        struct polylist *link;

};
typedef struct polylist *NODE;
#define MALLOC(p,s,t)\
        p=(t)malloc(s);\
        if(p==NULL)\
         {\
                  printf("insufficient memeory\n");\
                 exit;\
          }

//Prototypes
NODE readPoly(NODE);
NODE padd(NODE a,NODE b);
float evalPoly();
NODE attach(int, int, int, int,NODE);
 void printPoly(NODE);

//main function
void main()
{
        NODE p1,p2,p3;
        int done=0,
        choice; float result;
        int expx,expy,expz;
         while(!done)
        {
                printf("1.Polynomial Evaluation\t2.Add Polynomial\t3.Exit\n");
                printf("Enter your choice\n");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1:  result=evalPoly();
```

```
                                printf("The Polynomial Result=%8.2f\n",result);
                                break;
                        case 2: MALLOC(p1,sizeof(structpolylist),NODE);
                                MALLOC(p2,sizeof(struct polylist),NODE); p1->link=p1;
                                p2->link=p2;
                                printf("Enter polynomial a \n");
                                        p1=readPoly(p1);
                                printf("Enter polynomial b\n");
                                p2=readPoly(p2);
                                p3=padd(p1,p2);
                                printf("Resultant polynomial  c\n");
                                        printPoly(p3);
                                break;
                        case 3:  done=1;
                                break;
                        default:printf("Invalid Choice\n");
                                break;
                }
        }
}

// To read the polynomial- coeff, expx, expy, expz
 NODE readPoly(NODE p)
{
        int ce, expx, expy,expz;
        while(1)
        {
                printf("Enter the coeff and expx, expy,
                expz\n"); scanf("%d%d%d%d", &ce, &expx,
                &expy, &expz); if(expx==-1)
                break;
                p=attach(ce,expx,expy,exp
                z,p);
        }
        return p;
}
//To add the new polynomial
NODE padd(NODE p1, NODE p2)
{

        NODE p3;
        NODE starta=p1;
         NODE startb=p2;
        int sum;
        MALLOC(p3,sizeof(struct polylist),NODE);
        p3->link=p3;
        p1=p1->link;
        p2=p2->link;
        while(p1!=starta && p2!= startb)
        {
                if(p1->x==p2->x && p1->y==p2->y && p1->z==p2->z)
                 {
                        sum=p1->coef+p2->coef;
                        if(sum)
                        p3=attach(sum,p1->x,p1->y,p1->z,p3);
```

```
                        p1=p1->link;
                        p2=p2->link;
                    }
                    else if(p1->x<p2->x)
                    {
                        p3=attach(p2->coef,p2->x,p2->y,p2->z,p3);
                        p2=p2->link;
                    }
                    else if(p1->x>p2->x)
                    {
                        p3=attach(p1->coef,p1->x,p1->y,p1->z,p3);
                        p1=p1->link;
                    }
                    else if(p1->y<p2->y)
                    {
                        p3=attach(p2->coef,p2->x,p2->y,p2->z,p3);
                        p2=p2->link;
                    }
                    else if(p1->y>p2->y)
                    {
                        p3=attach(p1->coef,p1->x,p1->y,p1->z,p3);
                        p1=p1->link;
                    }
                    else if(p1->z<p2->z)
                    {
                        p3=attach(p2->coef, p2->x,p2->y,p2->z,p3);
                        p2=p2->link;
                    }
                    else if(p1->z>p2->z)
                    {
                        p3=attach(p1->coef,p1->x,p1->y,p1->z,p3);
                        p1=p1->link;
                    }
            }
            for(;p1!=starta;p1=p1->link)
            p3=attach(p1->coef,p1->x,p1->y,p1->z,p3);
            for(;p2!=startb;p2=p2->link)
            p3=attach(p2->coef,p2->x,p2->y,p2->z,p3);
            return p3;
}

//To attach the polynomial
NODE attach(int c, int expx, int expy, int expz, NODE header)
{
        NODE temp,q;
        MALLOC(temp,sizeof(struct polylist),NODE);
        temp->coef=c;
        temp->x=expx;
        temp->y=expy;
        temp->z=expz;
        q=header->link;
        while(q->link!=header)
        q=q->link;
        q->link=temp;
        temp->link=header;
```

```
        return header;
}

// To evaluate the polunomial
float evalPoly()
{
        float sum=0;
        NODE p, start;
        float
        xval,yval,zval;
        int k,n=20;
        int terms[20]={6,2,2,1,-4,0,1,5,3,3,1,1,2,1,5,1,-2,1,1,3};
        MALLOC(p,sizeof(struct
        polylist),NODE); p->link=p;
        printf("Enter values for x,y,z\n");
        scanf("%f%f%f", &xval, &yval,
        &zval); for(k=0;k<n;k+=4)
         p=attach(terms[k],terms[k+1],terms[k+2],terms[k+3],p);
        start=p;
        p=p-
        >link;
        do
         {
                sum+=p->coef*pow(xval,p->x)*pow(yval,p-
                >y)*pow(zval,p->z); p=p->link;
         }
         while(p!=s
         tart);
         return
         sum;
}
//To display the
polynomial void
printPoly(NODE header)
{
        NODE q;
        q=header-
        >link; do
        {
                printf("%dx^%dy^%dz^%d+", q->coef,q->x,q-
                >y,q->z); q=q->link;
         }
        while(q!=hea
        der);
        printf("\n");
}
```

**OUTPUT :**

```
[root@localhost 2022batchDSLab]# cc 9_ polynomial.c
 [root@localhost 2022batchDSLab]# ./a.out

 1.polynomial evaluation
 2.Add Polynomial
 3.Exit
 Enter your choice
 1
 Enter value for x,y,z 0 0 0
 The polynomial evaluation result = 0.00

 1.polynomial evaluation
 2.Add Polynomial
 3.Exit
 Enter your choice
 1
 Enter value for x,y,z 1 1 1
 The polynomial evaluation result =5.00

 1.polynomial evaluation
 2.Add Polynomial
 3.Exit
 Enter your choice
 2
 enter polynomial a
 enter the coeff and expx, expy, expz
 3 1 1 1
 enter the coeff and expx, expy, expz
 -1 -1 -1 -1
 enter polynomial b
 enter the coeff and expx,expy, expz
 4 1 1 1
 enter the coeff and expx, expy, expz
 -1 -1 -1 -1
 Resultant polynomial c
 7x^1y^1z^1
 +

 1.polynomial evaluation
 2.Add Polynomial
 3.Exit
 enter your
 choice
 2
 enter polynomial a
```

```
enter the coeff and expx, expy, expz
3 2 1 1
enter the coeff and expx, expy, expz
-1 -1 -1 -1
enter polynomial b
enter the coeff and expx, expy, expz
4 2 1 1
enter the coeff and expx, expy, expz
2 3 2 1
enter the coeff and expx, expy, expz
-1 -1 -1 -1
resultant polynomial c
7x^2y^1z^1+2x^3y^2z
^1
+

1.polynomial evaluation
2.Add       Polynomial
3.Exit
enter your
choice 2

enter polynomial a
enter the coeff and expx, expy, expz
4 5 1 2
enter the coeff and expx, expy, expz
-1 -1 -1 -1
enter polynomial b
enter the coeff and
expx, expy, expz 2 0
0 0
enter the coeff and expx, expy, expz
-1 -1 -1 -1
resultant polynomial c
4x^5y^1z^2+2x^0y^0z
^0
+

1.polynomial evaluation
2.Add  Polynomial
3.Exit
enter your
choice 3
```

**PROGRAM: 10**

**Develop 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. **Exit**

```
// Header files
#include<stdio.h>
#include<malloc.h>
#include<string.h>
#include<stdlib.h>
#define MAX 30

struct tree
{
        int info;
        struct tree *left;
        struct tree *right;
};
#define MALLOC(p,s,t)\
        p=(t)malloc(s);\
        if(p==NULL)\
        { \
                printf("insufficient memeory\n"); \
                 exit;\
        }
typedef struct tree *NODE;


//Prototypes
NODE create(NODE, int);
NODE createtree(NODE, int);
void Preorder(NODE);
void Postorder(NODE);
void Inorder(NODE);
 int search(NODE,int);
//Global Variable Declaration

int n;
//main function int main()
{
        int choice, done, flag, key; NODE p;
        p=NULL;
        done=0;
        while(!done)
        {
                printf("1.Create\t2.Preorder\t3.Inorder\t4.Postorder\t5.Search\t6.Exit\n");
                printf("Enter the choice\n");
```

```
                scanf("%d",&choice);
                switch (choice)
                {
                        case 1:  printf("Enter No. of Data Elements\n");
                                 scanf("%d",&n);
                                 p=create(p,n); break;

                        case 2:  Preorder(p);
                                 printf("\n");
                                 break;
                        case 3:  Inorder(p);
                                 printf("\n");
                                 break;
                        case 4:  Postorder(p);
                                 printf("\n");
                                 break;
                        case 5:  printf("Enter the Key to Search\n"); scanf("%d", &key);
                                 flag= search(p,key);
                                  if(flag==1)
                                 printf("key Found\n");
                                 else
                                 printf("key not Found\n");
                                 break;
                        case 6:  done=1;
                                 break;
                        default:  printf("Invalid Choice\n");
                }
        }
        return 0;
}

//Read function definition to create BST
NODE create(NODE root, int n)
{
      int i,e; NODE q;
      if(root==NULL)
      {
            for(i=1;i<=n;i++)
            {
                  printf("Enter Data element\n");
                  scanf("%d", &e);
                   root=createtree(root,e);
              }
              return root;
     }
    else
    printf("Tree already has some data\n");
  }
```

```
////Binary search tree creation function definition
NODE createtree(NODE p, int e)
{
     if(p==NULL)
     {
          MALLOC(p,sizeof(struct tree),NODE);
           p->info=e;
          p->left=p->right=NULL;
     }
    else if(e==p->info)
    printf("Duplicate Entry\n");
     else if(e<p->info)
    p->left=createtree(p->left,e);
    else
     p->right=createtree(p->right,e);
      return p;
}

//Search the key element in the BST
int search(NODE p, int key)
{
     if(p==NULL)
     return 0;
     else if(key==p->info)
      return 1;
     else if(key<p->info)
     return search(p->left,key);
     else
     return search(p->right,key);
}

//Preorder Traversal -VLR
 void Preorder(NODE p)
 {
     if(p!=NULL)
      {
          printf("%d\t", p->info);
          Preorder(p->left);
          Preorder(p->right);
      }
 }

//Inorder Traversal -LVR
void Inorder(NODE p)
{
     if(p!=NULL)
      {
          Inorder(p->left); printf("%d\t", p-
          >info); Inorder(p->right);
```

```
        }
}

//Postorder Traversal -LRV
    void Postorder(NODE p)
    {
        if(p!=NULL)
        {
                Postorder(p->left);
                Postorder(p->right);
                printf("%d\t", p->info);
        }
     }
```

**OUTPUT :**

```
[root@localhost 2022batchDSLab]# cc
10_BST.c [root@localhost
2022batchDSLab]# ./a.out

1.Create    2.Preorder    3.Inorder    4.Postorder    5.Search    6.Exit
 Enter the Choice
1
Enter No. of Data Elements 12
Enter Data Element 6
Enter Data Element 9
Enter Data Element 5
Enter Data Element 2
Enter Data Element 8
Enter Data Element 15
Enter Data Element 24
Enter Data Element 14
Enter Data Element 7
Enter Data Element 8
Duplicate Entry Enter Data Element 5
Duplicate Entry Enter Data Element 2
Duplicate Entry

1.Create    2.Preorder    3.Inorder    4.Postorder    5.Search    6.Exit
Enter the Choice
3

2  5    6    7    8    9    14    15    24

1.Create    2.Preorder    3.Inorder    4.Postorder    5.Search    6.Exit
Enter the Choice
2

6  5    2    9    8    7    15    14    24

1.Create    2.Preorder    3.Inorder    4.Postorder    5.Search    6.Exit
Enter the Choice
4
```

2 5 7 8 14 24 15 9 6

1.Create 2.Preorder 3.Inorder 4.Postorder 5.Search 6.Exit
Enter the Choice
5

Enter the Key to
Search 7
Key Found

1.Create 2.Preorder 3.Inorder 4.Postorder 5.Search 6.Exit
Enter the Choice
5

Enter the Key to
Search 30
Key not Found

1.Create 2.Preorder 3.Inorder 4.Postorder 5.Search 6.Exit
Enter the Choice
6

**Program: 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 diagraph using DFS/BFS method.**

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 10
int visited[MAX];
int a[MAX][MAX];                    //adjacency matrix
int n;                              //no. of vertices
void ReadAdjMatrix();
void dfs(int);
int  main()
{
        int start;
        ReadAdjMatrix();
        printf("enter the starting vertex\n");
        scanf("%d", &start);
        dfs(start);
        return 0;
}
void ReadAdjMatrix()
{
        int i,j;
        printf("enter the no. of vertices\n");
        scanf("%d", &n);
        printf("enter Adjacency matrix\n");
        for(i=1;i<=n;i++)
                for(j=1;j<=n;j++)
                scanf("%d", &a[i][j]);
}
void dfs(int v)
{
        int w;
        visited[v]=1;
        for(w=1;w<=n;w++)
        {
                if(visited[w]==0 &&a[v][w]==1)
                 {
printf("%d->",w);
                        dfs(w);
                 }
         }
         printf("\n");
```

```
}
```

**OUTPUT:**

```
[root@localhost 2022batchDSLab]# cc 11_matrix.c
[root@localhost 2022batchDSLab]# ./a.out

GRAPH TRAVERSALS USING ADJACENCY MATRIX

Enter the number of nodes
5
Enter the adjacency matrix 0 1 1 1 0
1 0 1 0 0
1 1 0 0 1
1 0 0 0 0
0 1 1 0 0

GRAPH TRAVERSALS USING ADJACENCY MATRIX

1:Reachable nodes using BFS
2:Reachable nodes using DFS
 3:Exit

Enter the Choice
 1
Enter the Source Node
2
The nodes visited from 2 0 1 4 3

GRAPH TRAVERSALS USING ADJACENCY MATRIX

1:Reachable nodes using BFS
 2:Reachable nodes using DFS
 3:Exit

Enter the Choice
2

 Reachable Node from 0:  0  1  2  4  3

 Reachable Node from 1:  1  0  2  4  3

 Reachable Node from 2:  2  0  1  3  4

 Reachable Node from 3:  3  0  1  2  4

 Reachable Node from 4:  4  1  0  2  3

GRAPH TRAVERSALS USING ADJACENCY MATRIX

1:Reachable nodes using BFS
 2:Reachable nodes using DFS
3:Exit
```

```
Enter the Choice
3

GRAPH TRAVERSALS USING ADJACENCY MATRIX

Enter the number of nodes 5
Enter the adjacency matrix 0 1 1 1 0
1 0 0 1 0
1 0 0 1 1
1 1 1 0 1
0 0 1 1 0

GRAPH TRAVERSALS USING ADJACENCY MATRIX

1:Reachable nodes using BFS
2:Reachable nodes using DFS
 3:Exit

Enter the Choice
 1
Enter the Source Node
 0
The nodes visited from 0 1 2 3 4

GRAPH TRAVERSALS USING ADJACENCY MATRIX

1:Reachable nodes using BFS
 2:Reachable nodes using DFS
3:Exit

Enter the Choice
2
        Reachable Node from 0:  0   1   3   2   4
        Reachable Node from 1:  1   0   2   3   4
        Reachable Node from 2:  2   0   1   3   4
        Reachable Node from 3:  3   0   1   2   4
        Reachable Node from 4:  4   2   0   1   3

GRAPH TRAVERSALS USING ADJACENCY MATRIX

1:Reachable nodes using BFS
2:Reachable nodes using DFS
 3:Exit

Enter the Choice
3
```

GRAPH TRAVERSALS USING ADJACENCY MATRIX

Enter the number of nodes 5
Enter the adjacency matrix 0 1 1 0 0
0 0 0 1 1
0 0 0 1 0
1 0 0 0 1
0 0 0 0 0

GRAPH TRAVERSALS USING ADJACENCY MATRIX

1:Reachable nodes using BFS
2:Reachable nodes using DFS
3:Exit

Enter the Choice
1
Enter the Source Node
2
The nodes visited from 2 3 0 4 1

GRAPH TRAVERSALS USING ADJACENCY MATRIX

1:Reachable nodes using BFS
2:Reachable nodes using DFS
3:Exit

Enter the Choice
2

Reachable Node from 0: 0 1 3 4 2

Reachable Node from 1: 1 3 0 2 4

Reachable Node from 2: 2 3 0 1 4

Reachable Node from 3: 3 0 1 4 2

Reachable Node from 4: 4

GRAPH TRAVERSALS USING ADJACENCY MATRIX

1:Reachable nodes using BFS

2:Reachable nodes using DFS

3:Exit
3

**PROGRAM: 12**

**Design and develop a program in C that uses Hash Function H:K->L as H(K)=K mod m(reminder method) and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.**

```
// Header files
#include<stdio.h>
#include<stdlib.h>
#define MAX 100

//Prototypes
int create(int);
void linearprobe(int[], int, int);
void Display(int []);

//main function
int  main()
{
	int a[MAX], num, key,i;
	int ans=1;
	printf("	Collision Handling by linear probing\n\n");
	for(i=0;i<MAX;i++)
	 a[i]=-1;
	do
	{
		printf("Enter the Employee ID\n");
		scanf("%4d", &num);
		key= create(num);
		linearprobe(a,key,num);
		printf(" Do you wish to continue adding one more employee id?(1/0)\n");
		scanf("%d", &ans);
	}
	while(ans);
	Display(a);
	 return 0;
}

//Create hash table function definition
int create(int num)
{
	int key;
	key=num%100;
	return key;
}

//linearprobing function definition
void linearprobe(int a[MAX], int key, int num)
{
	int flag=0, i, count=0;
	if(a[key]==-1)
	a[key]=num;
	else
```

```
	{
		printf("Collision Detected!\n");
		for(i=0;i<MAX;i++)
		 if(a[i]!=-1)
		 count++;
		 printf("Collision avoided successfully using LINEAR PROBING\n");
		if(count==MAX)
		{
			printf("Hash table is full\n");
			Display(a);
			exit(1);
		 }
		 for(i=key+1;i<MAX;i++)
		 if(a[i]==-1)
		{
			 a[i]=num;
			 flag=1;
			break;
		}

		 for(i=0;i<key && flag==0;i++)
		if(a[i]==-1)
		{
			 a[i]=num;
			 flag=1;
			 break;
		}
	}
}


//Display hash table function definition
void Display(int a[MAX])
{
	int i, choice;
	printf("\n1.Display All \n2. Filtered Display\n");
	scanf("%d", &choice);
	if(choice==1)
	{
		printf("Complete Hash Table is\n");
		for(i=0;i<MAX;i++)
		printf("\n%d\t%d",i,a[i]);
	 }
	else
	{
		printf("Filtered Hash Table is\n");
		for(i=0;i<MAX;i++)
		if(a[i]!=-1)
		 {
			printf("\n%d\t%d\n",i,a[i]);
			continue;
		 }
	 }
}
```

**OUTPUT :**

```
[root@localhost 2022batchDSLab]# cc 12_hashfunction.c
[root@localhost 2022batchDSLab]# ./a.out

Collision Handling by linear probing

Enter the Employee ID
101

Do you wish to continue adding one more employee id?(1/0)
1

Enter the Employee ID
201

Collision Detected!

Collision avoided successfully using LINEAR PROBING

Do you wish to continue adding one more employee id?(1/0)
1

Enter the Employee ID
500

Do you wish to continue adding one more employee id?(1/0)
1

Enter the Employee ID
406

Do you wish to continue adding one more employee id?(1/0)
1

Enter the Employee ID
303

Do you wish to continue adding one more employee id?(1/0)
1

Enter the Employee ID
403

Collision Detected!

Collision avoided successfully using LINEAR PROBING
```

Do you wish to continue adding one more employee id?(1/0)
1

Enter the Employee ID
404
Collision Detected!

Collision avoided successfully using LINEAR PROBING

Do you wish to continue adding one more employee id?(1/0)
1

Enter the Employee ID
505

Collision Detected!

Collision avoided successfully using LINEAR PROBING

Do you wish to continue adding one more employee id?(1/0)
0

1.Display All

2.Filtered Display

2

Filtered Hash Table is

0 500

1 101

2 201

3 303

4 403

5 404

6 406

7 505

**OUTPUT 2:**

```
[root@localhost 2022batchDSLab]# ./a.out

Collision Handling by linear probing
Enter the Employee ID
999

 Do you wish to continue adding one more employee id?(1/0)
1
Enter the Employee ID
555
 Do you wish to continue adding one more employee id?(1/0)
1
Enter the Employee ID
400
 Do you wish to continue adding one more employee id?(1/0)
1
Enter the Employee ID
500
Collision Detected!
Collision avoided successfully using LINEAR PROBING
 Do you wish to continue adding one more employee id?(1/0)
1
Enter the Employee ID
203
Do you wish to continue adding one more employee id?(1/0)
1
Enter the Employee ID
303
Collision Detected!
Collision avoided successfully using LINEAR PROBING
Do you wish to continue adding one more employee id?(1/0)
1
Enter the Employee ID
404
Collision Detected!
Collision avoided successfully using LINEAR PROBING
Do you wish to continue adding one more employee id?(1/0)
0
1.Display All
2. Filtered Display
1
```

```
Complete Hash Table is
0  400
1  500
2  -1
3  203
4  303
5  404
6  -1
7  -1
8  -1
9  -1
10 -1
11 -1
12 -1
13 -1
14 -1
15 -1
16 -1
17 -1
18 -1
19 -1
20 -1
21 -1
22 -1
23 -1
24 -1
25 -1
26 -1
27 -1
28 -1
29 -1
30 -1
31 -1
32 -1
33 -1
34 -1
35 -1
36 -1
37 -1
38 -1
39 -1
40 -1
41 -1
42 -1
43 -1
44 -1
45 -1
46 -1
47 -1
48 -1
49 -1
50 -1
51 -1
```

52 -1
53 -1
54 -1
55 555
56 -1
57 -1
58 -1
59 -1
60 -1
61 -1
62 -1
63 -1
64 -1
65 -1
66 -1
67 -1
68 -1
69 -1
70 -1
71 -1
72 -1
73 -1
74 -1
75 -1
76 -1
77 -1
78 -1
79 -1
80 -1
81 -1
82 -1
83 -1
84 -1
85 -1
86 -1
87 -1
88 -1
89 -1
90 -1
91 -1
92 -1
93 -1
94 -1
95 -1
96 -1
97 -1
98 -1
99 999