EAST POINT COLLEGE OF ENGINEERING & TECHNOLOGY

Department of Computer Science and Engineering

'Jnana Prabha', Virgo Nagar Post, Bengaluru-560049

Academic Year: 2023-24

LABORATORY MANUAL

Semester : **III**

Subject : **Data Structures Laboratory**

Subject Code : **BCSL305**

NAME: _____

USN: _____

SECTION: _____

PROGRAM OUTCOMES

Engineering Graduates will able to

Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

Problem analysis: Identify, formulate, review research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

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.

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.

Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activitieswith an understanding of the limitations.

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.

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.

Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and writeeffective reports and design documentation, make effective presentations, and give and receive clear instructions.

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.

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.

EAST POINT COLLEGE OF ENGINEERING & TECHNOLOGY

Department of Computer Science and Engineering

INSTITUTE VISION AND MISSION

VISION

The East Point College of Engineering and Technology aspires to be a globally acclaimed institution, recognized for excellence in engineering education, applied research and nurturing students for holistic development.

MISSION

M1: To create engineering graduates through quality education and to nurtureinnovation, creativity and excellence in teaching, learning and research

M2: To serve the technical, scientific, economic and societal developmental needsof our communities

M3: To induce integrity, teamwork, critical thinking, personality development andethics in students and to lay the foundation for lifelong learning

EAST POINT COLLEGE OF ENGINEERING & TECHNOLOGY

Department of Computer Science and Engineering

DEPARTMENT VISION AND MISSION

VISION

The department aspires to be a Centre of excellence in Computer Science & Engineering to develop competent professionals through holistic development.

MISSION

M1: To create successful Computer Science Engineering graduates through effective pedagogies, the latest tools and technologies, and excellence in teaching and learning.

M2: To augment experiential learning skills to serve technical, scientific, economic, and social developmental needs.

M3: To instil integrity, critical thinking, personality development, and ethics in students for a successful career in Industries, Research, and Entrepreneurship.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO 1: To produce graduates who can perform technical roles to contribute effectively in software industries and R&D Centre

.

PEO 2: To produce graduates having the ability to adapt and contribute in key domains of computer science and engineering to develop competent solutions.

PEO 3: To produce graduates who can provide socially and ethically responsible solutions while adapting to new trends in the domain to carve a successful career in the industry

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO1: To conceptualize, model, design, simulate, analyse, develop, test, and validate computing systems and solve technical problems arising in the field of computer science & engineering.

PSO2: To specialize in the sub-areas of computer science & engineering systems such as cloud computing, Robotic Process Automation, cyber security, big data analytics, user interface design, and IOT to meet industry requirements.

PSO3: To build innovative solutions to meet the demands of the industry using appropriate tools and techniques.

COURSE LEARNING OBJECTIVES

This laboratory course enables students to get practical experience in design, develop, implement, analyze and evaluation/testing of

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

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

CO 1. Identify different data structures and their applications.

CO 2. Apply stack and queues in solving problems.

CO 3. Demonstrate applications of linked list.

CO 4. Explore the applications of trees and graphs to model and solve the real-world problem

CO 5. Make use of hashing techniques and resolve collisions during mapping of key value pairs

DATA STRUCTURES AND APPLICATIONS

[As per Choice Based Credit System (CBCS) scheme]
(Effective from the academic year 2023-2024)

SEMESTER – III

Course Code	**BCSL305**	**CIE Marks**	**50**
Number of Lecture Hours/Week	**0:0:2**	**SEE Marks**	**50**
Total Number of Lecture Hours	**28**		

CREDITS – 01

Course Learning Objectives:

This laboratory course enables students to get practical experience in design, develop, implement, analyze and evaluation/testing of

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

	Laboratory Component Description
	Prerequisite
	• Students should be familiarized about C Programming Concepts • Familiar with Linux.
Sl.No	***List of problems for which student should develop program and execute in the Laboratory***
1	**Aim**: Demonstrate Creation and Display of array of structures **Program**: 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.
2	**Aim**: Demonstrate the string Operations **Program**: 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.
3	**Aim**: Implement stack operations **Program**: Develop a menu driven Program in C for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX) a. Push an Element on to Stack b. Pop an Element from Stack c. Demonstrate how Stack can be used to check Palindrome d. Demonstrate Overflow and Underflow situations on Stack e. Display the status of Stack f. Exit Support the program with appropriate functions for each of the above operations
4	**Aim**: Implement the applications of stack

	Program: 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
5	**Aim**: Implement the applications of stack **Program**: Design, Develop and Implement a Program in C for the following Stack Application a. Evaluation of Suffix expression with single digit operands and operators: +, -, *, /, %, ^ b. Solving Tower of Hanoi problem with n disks
6	**Aim**: Demonstrate the implementation of Circular Queue **Program**: 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
7	**Aim**: Implement SLL Single linked list (SLL) of N Students Data **Program**: 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
8	**Aim**: Implement stack and queue using doubly linked list(DLL) **Program**: 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
9	**Aim**: Construct Circular Single Linked List **Program**: Develop a Program in C for the following operations on Singly Circular Linked List (SCLL) with header nodes a. Represent and Evaluate a Polynomial $P(x,y,z) = 6x^2y^2z-4yz^5+3x^3yz+2xy^5z-2xyz^3$ 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	**Aim**: Construct Binary Search Tree **Program**: 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
11	**Aim**: Implement Graph Traversal Methods **Program**: 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
12	**Aim**: Demonstrate collision resolving **Program**: 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.

Course Outcomes

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

CO 1. Identify different data structures and their applications.
CO 2. Apply stack and queues in solving problems.
CO 3. Demonstrate applications of linked list.
CO 4. Explore the applications of trees and graphs to model and solve the real-world problem.
CO 5. Make use of Hashing techniques and resolve collisions during mapping of key value pairs

Suggested Learning Resources:

Textbooks:

1. Ellis Horowitz and Sartaj Sahni, Fundamentals of Data Structures in C, 2nd Ed, Universities Press, 2014.

Reference Books:

1. Seymour Lipschutz, Data Structures Schaum's Outlines, Revised 1st Ed, McGraw Hill, 2014.
2. Reema Thareja, Data Structures using C, 3rd Ed, Oxford press, 2012
3. Gilberg and Forouzan, Data Structures: A Pseudo-code approach with C, 2nd Ed, Cengage Learning,2014.
4. Jean-Paul Tremblay & Paul G. Sorenson, An Introduction to Data Structures with Applications,2nd Ed, McGraw Hill, 2013
5. A M Tenenbaum, Data Structures using C, PHI, 1989
6. Robert Kruse, Data Structures and Program Design in C, 2nd Ed, PHI, 1996.

Weblinks and Video Lectures (e-Resources):

- http://elearning.vtu.ac.in/econtent/courses/video/CSE/06CS35.html
- https://nptel.ac.in/courses/106/105/106105171/
- http://www.nptelvideos.in/2012/11/data-structures-and-algorithms.html
- https://www.youtube.com/watch?v=3Xo6P_V-qns&t=201s
- https://ds2-iiith.vlabs.ac.in/exp/selection-sort/index.html
- https://nptel.ac.in/courses/106/102/106102064/
- https://ds1-iiith.vlabs.ac.in/exp/stacks-queues/index.html
- https://ds1-iiith.vlabs.ac.in/exp/linked-list/basics/overview.html
- https://ds1-iiith.vlabs.ac.in/List%20of%20experiments.html
- https://ds1-iiith.vlabs.ac.in/exp/tree-traversal/index.html
- https://ds1-iiith.vlabs.ac.in/exp/tree-traversal/depth-first-traversal/dft-practice.html
- https://infyspringboard.onwingspan.com/web/en/app/toc/lex_auth_013501595428077568125 59/overview

Conduct of Practical Examination:

- Experiment distribution
  Students are allowed to pick one experiment from the lot with equal opportunity.

- Change of experiment is allowed only once and marks allotted for procedure to be made zero of the changed part only.

- Marks Distribution (Need to change in accordance with university regulations)
  For laboratories having only one part – Procedure + Execution + Viva-Voce:
  15+70+15 = 100 Marks

Index					
Sl No	**Program List**	**CO**	**PO**	**RBT**	**Page No**
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.	CO1	PO3	L3	14
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.	CO1	PO3	L3	19
3	Design, Develop and Implement a menu driven Program in C for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX) a. Push an Element on to Stack b. Pop an Element from Stack c. Demonstrate how Stack can be used to check Palindrome d. Demonstrate Overflow and Underflow situations on Stack e. Display the status of Stack f. Exit Support the program with appropriate functions for each of the above operations	CO1	PO3	L3	23
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	CO1	PO3	L3	29
5	Design, Develop and Implement a Program in C for the following Stack Applications a. Evaluation of Suffix expression with single digit operands and operators: +, -, *, /, %, ^ b. Solving Tower of Hanoi problem with n disks	CO1	PO3	L3	33
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	CO2	PO3	L3	36

	c. Demonstrate Overflow and Underflow situations on Circular QUEUE d. Display the status of Circular QUEUE e. Exit Support the program with appropriate functions for each of the above operations				
7	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, Ph No 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	CO2	PO3	L3	41
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	CO3	PO3	L3	46
9	Develop a Program in C for the following operations on Singly Circular Linked List (SCLL) with header nodes a. Represent and Evaluate a Polynomial $P(x,y,z) = 6x^2y^2z-4yz^5+3x^3yz+2xy^5z-2xyz^3$ 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	CO3	PO3	L3	51
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	CO3	PO3	L3	59
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	CO4	PO3	L3	66

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.	CO5	PO3	L3	74

Course Articulation Matrix

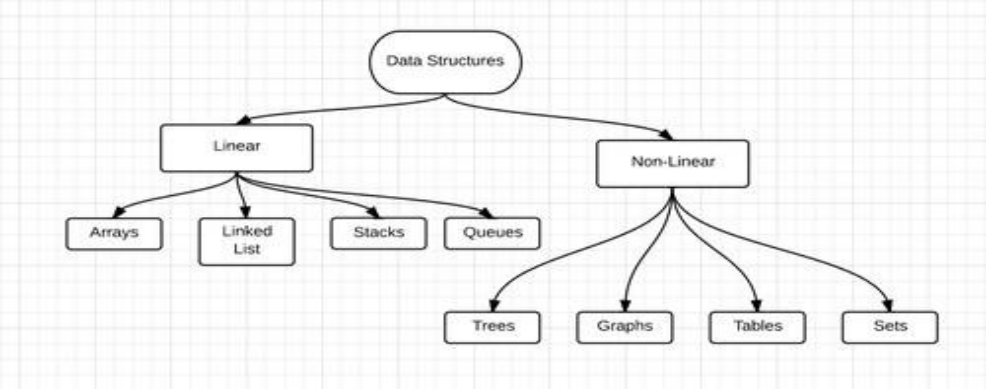
COs	Pos												PSOs		
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	3	3	3										3		
CO2	3	3	3									1	3		
CO3	3	3	3									1	3		
CO4	3	3	3									1	3		
CO5	3	3	3									1	3		

3 - High Correlation **2 - Medium Correlation** **1 – Low Correlation**

INTRODUCTION TO DATA STRUCTURES

Data Structure is defined as the way in which data is organized in the memory

location.There are 2 types of data structures:



Linear Data Structure:
In linear data structure all the data are stored linearly or contiguously in the memory. All the data are saved in continuously memory locations and hence all data elements are saved in one boundary. A linear data structure isone in which we can reach directly only one element from another while travelling sequentially. The main advantage, we find the first element, and then it's easy to find the next data elements. The disadvantage, the sizeof the array must be known before allocating the memory.

The different types of linear data structures are:

- Array
- Stack
- Queue
- Linked List

Non-Linear Data Structure:
Every data item is attached to several other data items in a way that is specific for reflecting relationships. Thedata items are not arranged in a sequential structure.

The various types of non linear data structures are:

- Trees
- Graphs

EXPERIMENT - 01

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.

ABOUT THE EXPERIMENT:

An **Array** is a collection of similar / same elements. In this experiment the array can be represented as one / singledimensionalelements.

Menu driven program in c - language to perform various array operations are implemented with the help of user definedfunctions as followings;

a. create()
b. display()
c. exit()

ALGORITHM:

Step 1: Create a structure to represent day.
Step 2: Read data for the calendar.
Step 3: Display the weekly activity report
Step 4: Stop.

PROGRAM CODE:

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

// Structure to represent a day in the calendar
struct Day
{
        char *name;
         int date;
        char *activity;
};

// Function to create a day in the calendar
struct Day create()
{
    struct Day day;
    day.name = (char *)malloc(20 * sizeof(char)); // Allocating memory for the name
    day.activity = (char *)malloc(100 * sizeof(char)); // Allocating memory for the activity
    printf("Enter the day name: ");
    scanf("%s", day.name);
    printf("Enter the date: ");
    scanf("%d", &day.date);
    printf("Enter the activity for the day: ");
    scanf(" %[^\n]", day.activity);
    return day;
}
```

```
// Function to read data for the calendar
void read(struct Day calendar[], int size)
{
        for (int i = 0; i < size; i++)
{
        calendar[i] = create();
}
}

// Function to display the weekly activity details report
void display(struct Day calendar[], int size)
{
        printf("\nWeekly Activity Details:\n");
        for (int i = 0; i < size; i++)
        {
                printf("Day %d: %s\n", i + 1, calendar[i].name);
                printf("Date: %d\n", calendar[i].date);
                printf("Activity: %s\n", calendar[i].activity);
                printf("\n");
        }
}

int main()
{
  int weekSize = 7;
  struct Day calendar[weekSize];
// Create the calendar
  read(calendar, weekSize);
// Display the weekly activity details
  display(calendar, weekSize);
// Free dynamically allocated memory
   for (int i = 0; i < weekSize; i++)
     {
       free(calendar[i].name);
       free(calendar[i].activity);
     }
   return 0;
}
```

Sample Output

```
Enter the day name: Monday
Enter the date: 161023
Enter the activity for the day: Reading
Enter the day name: Tuesday
Enter the date: 171023
Enter the activity for the day: Coding
Enter the day name: Wednesday
Enter the date: 181023
Enter the activity for the day: Painting
Enter the day name: Thursday
Enter the date: 191023
```

Enter the activity for the day: playing
Enter the day name: Friday
Enter the date: 201023

Enter the activity for the day: shopping
Enter the day name: Saturday
Enter the date: 211023
Enter the activity for the day: Watching Movie
Enter the day name: Sunday
Enter the date: 221023
Enter the activity for the day: Completing Assignments

Weekly Activity Details:
Day 1: Monday
Date: 161023Activity: Reading

Day 2: Tuesday
Date: 171023
Activity: Coding

Day 3: Wednesday
Date: 181023
Activity: Painting

Day 4: Thursday
Date: 191023
Activity: playing

Day 5: Friday
Date: 201023
Activity: shopping

Day 6: Saturday
Date: 211023
Activity: Watching Movie

Day 7: Sunday
Date: 221023
Activity: Completing Assignments

EXPERIMENT - 02

Design, Develop and Implement a program in C for the following operations on Strings

a. Read a Main String (STR), a Pattern String (PAT) and a Replace String (REP).
b. Perform Pattern Matching Operation: Find and Replace all occurrences of PAT in STR with REP if PATexists in STR. Repost 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.

ABOUT THE EXPERIMENT:

Strings are actually one-dimensional array of characters terminated by a **null** character '\0'. Thus a null- terminated string contains the characters that comprise the string followed by a **null**.

The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello."

char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

If you follow the rule of array initialization then you can write the above statement as follows:

char greeting[] = "Hello";

C language supports a wide range of built-in functions that manipulate null-terminated strings as follows:

strcpy(s1, s2); Copies string s2 into string s1.
strcat(s1, s2); Concatenates string s2 onto the end
of string s1.strlen(s1); returns the length of string s1.
strcmp(s1, s2); Returns 0 if s1 and s2 are the same; less than 0 if s1<s2;
greater
than 0 if s1>s2.S
strchr(s1, ch); Returns a pointer to the first occurrence of character ch in
string s1. strstr(s1, s2); Returns a pointer to the first occurrence of string s2 in
string s1.

ALGORITHM:

Step 1: Start.
Step 2: Read main string STR, pattern string PAT and replace
string REP.Step 3: Search / find the pattern string PAT in the
main string STR.
Step 4: if PAT is found then replace all occurrences of PAT in main string STR with
REP string.Step 5: if PAT is not found give a suitable error message.
Step 6: Stop.

PROGRAM CODE:

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

//Declarations
 char str[100], pat[50],
```

```
rep[50], ans[100];int i, j, c,

m, k, flag=0;

void stringmatch()
{
        i = m = c = j = 0;

        while(str[c] ! = '\0')
        {
                if(str[m] = = pat[i])      //..... matching
                {
                        i++; m++;
                        if(pat[i] = = '\0')          // ....found occurrences.
                        {
                                flag = 1;
                                // ....copy replace string in ans string.
                                for(k = 0; rep[k] != '\0'; k++, j++)
                                   ans[j] = rep[k];
                                   i = 0;
                                   c = m;
                        }
                }       // if ends.
                else            //... mismatch
                {
                     ans[j] = str[c];
                     j++; c++;
                     m = c; i = 0;
                  }//else ends
        } //end of while ans[j] = '\0';
  } //end stringmatch()

            void main()
            {
                    clrscr();
                    printf("\nEnter a main string \n"); ]
                    gets(str);
                    printf("\nEnter a pattern string \n");
                    flushall();
                    gets(pat);
                    printf("\nEnter a replace string \n");
                    flushall();
                    gets(rep);
                    stringmatch();
                    if(flag = = 1)
                            printf("\nThe resultant string is\n %s" , ans);
                            else
                            printf("\nPattern string NOT found\n");
                    getch();
            }       // end of main
```

SAMPLE OUTPUT:

RUN 1:
Enter a main string
Test
Enter a pattern string

Te
Enter a replace string
Re

The resultant string is
Rest

RUN 2:
Enter a main string
This is Data Structure lab
Enter a pattern string
Data Structure
Enter a replace string
Data structure with C

The resultant string is
This is Data structure with C lab

EXPERIMENT - 03

Design, Develop and Implement a menu driven program in C for the following operations on **STACK** of integers (Array implementation of stack with maximum size **MAX**)

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

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

ABOUT THE EXPERIMENT:

A stack is an abstract data type (ADT), commonly used in most programming languages. It is named stack as it behaves like a real-world stack, for example − deck of cards or pile of plates etc.


A real-world stack allows operations at one end only. For example, we can place or remove a card or plate from top of the stack only. Likewise, Stack ADT allows all data operations at one end only. At any given time, we can only access the top element of a stack.

This feature makes it LIFO data structure. LIFO stands for Last-in-first-out. Here, the element which is placed (inserted or added) last is accessed first. In stack terminology, insertion operation is called **PUSH** operation and removal operation is called **POP** operation.

Below given diagram tries to depict a stack and its operations −

A stack can be implemented by means of Array, Structure, Pointer and Linked-List. Stack can either bea fixed size one or it may have a sense of dynamic resizing. Here, we are going to implement stack using arrays which makes it a fixed size stack implementation.

Basic Operations:

- **push()** - pushing (storing) an element on the stack.
- **pop()** - removing (accessing) an element from the stack.

To use a stack efficiently we need to check status of stack as well. For the same purpose, the following functionality is added to stacks;

- . **isFull()** − check if stack is full.

- **isEmpty()**-check if stack is empty

ALGORITHM:

Step 1: Start.
Step 2: Initialize stack size MAX and top of stack -1.
Step 3: Push integer element on to stack and display the contents of the stack.if stack is full give a message as 'Stack is Overflow'.
Step 4: Pop element from stack along with display the stack contents.if stack is empty give a message as 'Stack is Underflow'.
Step 5: Check whether the stack contents are Palindrome or not.
Step 6: Stop.

PROGRAM CODE

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
#define max 5
int st[max],top=-1;
void push(),disp(),palin();
int pop();
void main()
{
int ch,k,item;clrscr();
while(1)
{
        printf("MAIN MENU\n");
        printf(" 1:Push\n 2:Pop\n 3:Display\n 4:Palindrome\n 5:Exit\n");
        printf("Enter your choice\n");
        scanf("%d",&ch);
        switch(ch)
                {
                        case 1:printf("Enter an item to push\n");
                                scanf("%d",&item);
                                push(item);
                                break;
                        case 2: k=pop();
                                if(k)
                                printf("popped element is %d\n",k);
                                break;
                        case 3:disp();
                                break;
                        case 4:palin();
                                break;
                        case 5:exit(0);
                }
        }
}
void push(int item)
{
        if(top==max-1)
```

```
        {
                printf("Stack overflow\n");
                return ;
        }



        st[++top]=item;
}
int pop()
{
        if(top==-1)
        {
        printf("Stack underflow\n");return 0;
        }
        return(st[top--]);
}
void palin()
{
        int i;
        char p[100];
        top=-1;
        printf("Enter a number to check for palindrome\n");
        fflush(stdin);
        gets(p);
        for(i=0;i<strlen(p);i++)
        push(p[i]-'0');
        for(i=0;i<strlen(p)/2;i++)
        if((p[i]-'0')!=pop())
                {
                        printf("the number is not palindrome\n");
                        return;
                }
                        printf("the number is palindrome\n");

}
void disp()
{
   int i;
   if(top==-1)
        {
                printf("Stack Empty\n");
                return;
        }
          printf("the stack contents are");
          for(i=top;i>=0;i--)
          printf("|%d|\n",st[i]);
        }
   }
```

SAMPLE OUTPUT:

```
----MAIN MENU----
 1.PUSH
```

2 POP
3.DISPLAY
4.PALINDROME
5.EXIT

Enter Your Choice: 1
Enter an element to be pushed: 2
(-------- AFTER THE 4 TIMES PUSH OPERATION)

----MAIN MENU----
PUSH
POP
3.DISPLAY
4.EXIT

Enter Your Choice: 1
Enter an element to be pushed: 9
Stack is Overflow

----MAIN MENU----
1. PUSH
2. POP
3.DISPLAY
4. PALINDROME
4. EXIT
Ener Your Choice: 3
Stack contents are
|4|
|3|
|2|
|1|

----MAIN MENU----
1. PUSH
2. POP
3.DISPLAY
4. PALINDROME
4. EXIT
Ener Your Choice: 2
Poped element is: 4

----MAIN MENU----
1. PUSH
2. POP
3.DISPLAY
4. PALINDROME
4. EXIT
Enter Your Choice: 2
Stack is Underflow

----MAIN MENU----
1.PUSH
2.POP

3.DISPLAY
4.PALINDROME
5.EXIT
Enter Your Choice: 4
Enter the number to check for palindrome:121
The number is palindrome

---MAIN MENU----

1. PUSH
2. POP
3.DISPLAY
3. PALINDROME
4. EXIT
Enter Your Choice: 4
Enter the number to check for palindrome:123
The number is not palindrome

EXPERIMENT - 04

Design, Develop and Implement a Program in C for converting an Infix Expression to Postfix Expression. Program should support for both parenthesized and free parenthesized expressions with the operators: **+, -, *, /, %(Remainder), ^ (Power)** and **alphanumeric** operands**.**

ABOUT THE EXPERIMENT:

Infix: Operators are written in-between their operands. Ex: X + Y
Prefix: Operators are written before their operands. Ex: +X Y
postfix: Operators are written after their operands. Ex: XY+

Examples of Infix, Prefix, and Postfix

Infix Expression	Prefix Expression	Postfix Expression
A + B	+ A B	A B +
A + B * C	+ A * B C	A B C * +

Convert Infix expression to postfix form: (A+B^D)/(E-F)+G

Symbol Scanned	STACK	Expression P
	(	
(	((	
A	((	A
+	((+	A
B	((+	A B
^	((+ ^	A B
D	((+ ^	A B D
)	(	A B D ^ +
/	(/	A B D ^ +
(	(/ (	A B D ^ +
E	(/ (	A B D ^ + E
-	(/ (-	A B D ^ + E
F	(/ (-	A B D ^ + E
)	(/	A B D ^ + E -
+	(+	A B D ^ + E - /
G	(+	A B D ^ + E - / G
)		A B D ^ + E - / G +

ALGORITHM:

Step 1: Start.
Step 2: Read an infix expression with parenthesis and without parenthesis.
Step 3: convert the infix expression to postfix expression.
Step 4: Stop

PROGRAM CODE:

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

int F(char symbol)
{
        switch(symbol)
        {
                case '+' :
                case '-':           return 2;

                case '*':
                case '/':           return 4;

                case '^':
                case '$':           return 5;

                case '(':           return 0;

                case '#':           return -1;

                default: return 8;
        }
int G(char symbol)
{
        switch(symbol)
        {
                case '+':
                case '-':           return 1;

                case '*':
                case '/':           return 3;

                case '^':
                case '$':           return 6;

                case '(':           return 9;

                case ')':           return 0;

                default:            return 7;
      }
}
```

```
void infix_postfix(char infix[], char postfix[])
{
   int top, j, i;
   char s[30], symbol;top = -1;
   s[++top] = '#';
    j = 0;
    for(i=0; i < strlen(infix); i++)
      {
          symbol = infix[i];
          while(F(s[top]) > G(symbol))
          {
              postfix[j] = s[top--];j++;
          }
          if(F(s[top]) != G(symbol))
                    s[++top] = symbol;
           else
                    top--;
       }
        while(s[top] != '#')
          {
                postfix[j++] = s[top--];
          }
      postfix[j] = '\0';
  }

void main()
{
        char infix[20], postfix[20];
        clrscr();
        printf("\nEnter a valid infix expression\n");
        flushall();
        gets(infix); i
        nfix_postfix(infix,postfix);
        printf("\nThe infix expression is:\n");
        printf ("%s",infix);
        printf("\nThe postfix expression is:\n");
        printf ("%s",postfix);
        getch();
}
```

SAMPLE OUTPUT:

Enter a valid infix expression
(a+(b-c)*d)
The infix expression is:
(a+(b-c)*d)
The postfix expression is:
abc-d*+

EXPERIMENT - 05

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

ABOUT THE EXPERIMENT:

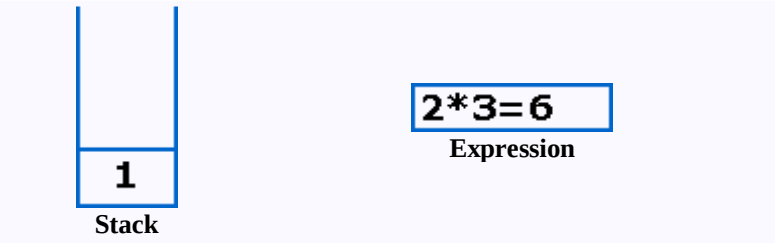
a. Evaluation of Suffix expression with single digit operands and operators: +, -, *, /, %, ^

Postfix/Suffix Expression: Operators are written after their operands. Ex: XY+
In normal algebra we use the infix notation like **a+b*c**. The corresponding postfix notation is **abc*+**

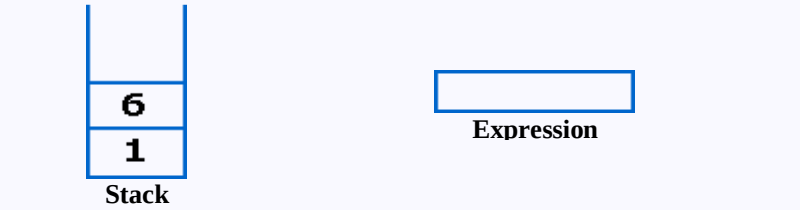
Example: **Postfix String: 123*+4-**

Initially the Stack is empty. Now, the first three characters scanned are 1,2 and 3, which are operands. Thus they will be pushed into the stack in that order.

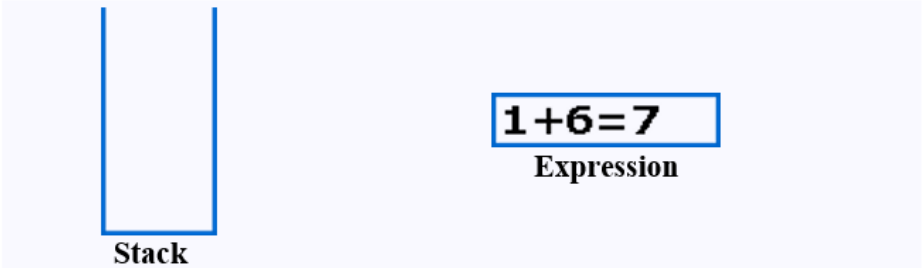
Next character scanned is "*", which is an operator. Thus, we pop the top two elements from the stack and perform the "*" operation with the two operands. The second operand will be the first element that is popped.



The value of the expression(2*3) that has been evaluated(6) is pushed into the stack.

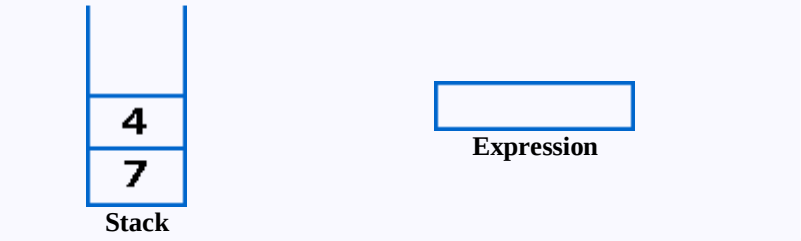


Next character scanned is "+", which is an operator. Thus, we pop the top two elements from the stack and perform the "+" operation with the two operands. The second operand will be the first element that is popped.

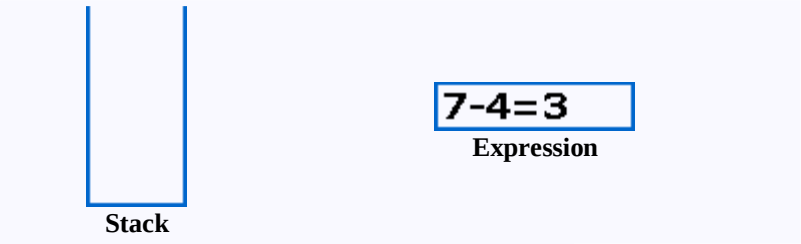


The value of the expression(1+6) that has been evaluated(7) is pushed into the stack.

Next character scanned is "4", which is added to the stack.



Next character scanned is "-", which is an operator. Thus, we pop the top two elements from the stack and perform the "-" operation with the two operands. The second operand will be the first element that is popped.



The value of the expression(7-4) that has been evaluated(3) is pushed into the stack.

Now, since all the characters are scanned, the remaining element in the stack (there will be only one element in the stack) will be returned.

End result:

Postfix String: 123*+4-
Result: 3

ALGORITHM:

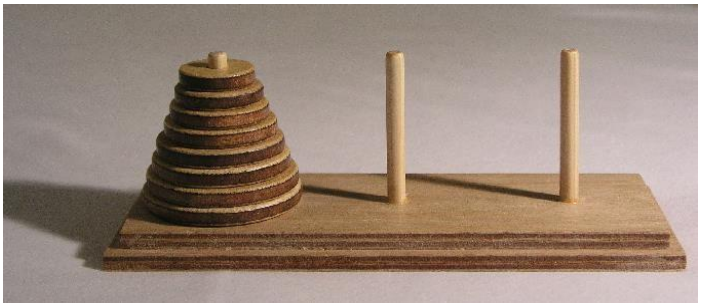
Step 1: Start.
Step 2: Read the postfix/suffix expression.
Step 3: Evaluate the postfix expression based on the precedence of the operator.
Step 4: Stop.

The **Tower of Hanoi** is a mathematical game or puzzle. It consists of three rods, and a number of disks of different sizes which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape.

The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

- Only one disk can be moved at a time.
- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
- No disk may be placed on top of a smaller disk.

With three disks, the puzzle can be solved in seven moves. The minimum number of moves required to solve a Tower of Hanoi puzzle is 2^n - 1, where *n* is the number of disks.



ALGORITHM:

Step 1: Start.
Step 2: Read N number of discs.
Step 3: Move all the discs from source to destination by using temp rod.
Step 4: Stop.

PROGRAM CODE:

PROGRAM 4A:

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

double compute(char symbol, double op1, double op2)
{
        switch(symbol)
        {
                case '+':       return op1 + op2;
                case' -':       return op1 - op2;
                case '*':       return op1 * op2;
                case '/':       return op1 / op2;
                case '$':
                case '^':      return pow(op1,op2);
                default:        return 0;
        }
}
void main()
{

        double s[20], res, op1, op2;
        int top, i;
```

```
        char postfix[20], symbol;
        clrscr();
        printf("\nEnter the postfix expression:\n");
        flushall();
        gets(postfix);
        top=-1;
        for(i=0; <strlen(postfix); i++)
           {
                symbol = postfix[i];
                if(isdigit(symbol))
                s[++top] = symbol - '0';
             else

              {             op2 = s[top--];
                            op1 = s[top--];
                            res = compute(symbol, op1, op2);
                            s[++top] = res;
               }
            }

      res = s[top--];
printf("\nThe result is : %f\n", res);
getch();
}
```

SAMPLE OUTPUT:

RUN1:

Enter the postfix expression:23+

The result is: 5.000000

RUN2:
Enter the postfix expression:23+7*

The result is: 35.000000

PROGRAM 5B:

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

void tower(int n, int source, int temp, int destination)
{
        if(n == 0)
         return;
         tower(n-1, source, destination, temp);
         printf("\nMove disc %d from %c to %c", n, source, destination);
          tower(n-1, temp, source, destination);
}

void main()
{
         int n; clrscr();
         printf("\nEnter the number of discs:  \n");
         scanf("%d", &n);
         tower(n, 'A', 'B', 'C');
         printf("\n\nTotal Number of moves are: %d", (int)pow(2,n)-1);
         getch();
}
```

SAMPLE OUTPUT:

```
Enter the number of discs:3
Move disc 1 from A to C
Move disc 2 from A to B
Move disc 1 from C to B
Move disc 3 from A to C
Move disc 1 from B to A
Move disc 2 from B to C
Move disc 1 from A to C

Total Number of moves are:  7
```

EXPERIMENT - 06

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

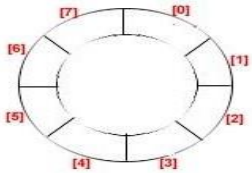
a. Insert an Element on to Circular QUEUE
b. Delete an Element from Circular QUEUE
c. Demonstrate ***Overflow*** and ***Underflow*** situations on Circular QUEUE
d. Display the status of Circular QUEUE
e. Exit

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

ABOUT THE EXPERIMENT:

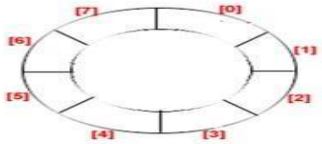
Circular queue is a linear data structure. It follows FIFO principle. In **circular queue** the last node is connected back to the first node to make a **circle**. **Circular** linked list fallow the First In First Out principle. Elements are added at the rear end and the elements are deleted at front end of the **queue**. The queue is considered as a circular queue when the positions 0 and MAX-1 are adjacent. Any position before front is also after rear.

A circular queue looks like

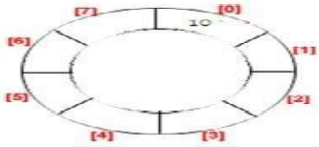


Consider the example with Circular Queue implementation:

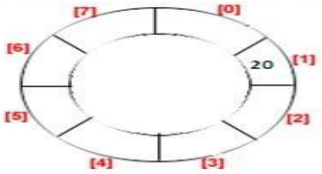
1) Initially: **Front = 0** and **rear = -1**



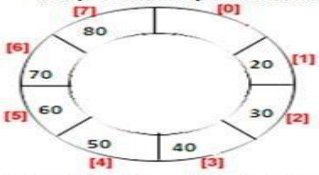
2) Add item 10 then **front = 0** and **rear =0**.



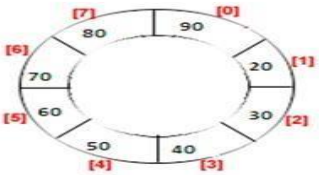
3) Now delete one item then **front = 1** and **rear =1**.



4) Like this now insert 30, 40, and 50,50,70,80 respectability then **front =1** and **rear = 7.**



5) Now in case of linear queue, we can not access 0 block for insertion but in circular queue next item will be inserted of 0 block then **front =0** and **rear = 0**.



ALGORITHM:

Step 1: Start.
Step 2: Initialize queue size to MAX.
Step 3: Insert the elements into circular queue. If queue is full give a message as 'queue is overflow"
Step 4: Delete an element from the circular queue. If queue is empty give a message as 'queue is underflow'.Step 5: Display the contents of the queue.
Step 6: Stop.

PROGRAM CODE:

```
#include<stdio.h>
#include<conio.h>
#define MAX 4
int ch, front = 0, rear = -1, count=0;
char q[MAX], item;

void insert(char item)
{
    if(count == MAX)
    printf("\nQueue is Full");
    return;
    else
    {
                rear = (rear + 1) % MAX;
                q[rear]=item;
                count++;

    }
}

  void del()
{
    if(count == 0)
    printf("\nQueue is Empty");
    return;
    else
    {
        if(front > rear && rear==MAX-1)
                {
                        front=0;        rear=-1;count=0;
                 }
          else
             {
                item=q[front];
                printf("\nDeleted item is: %c",item);
                front = (front + 1) % MAX;
                count--;
           }
    }
}
```

```
void display()
{
        int i, f=front, r=rear;
        if(count == 0)
                printf("\nQueue is Empty");
        else
{
                        printf("\nContents of Queue is:\n");
                        for(i=0; i<=count; i++)
                        {
                                printf("%c\t", q[f]);
                                 f = (f + 1) % MAX;
                        }
                }
}

void main()
{
        clrscr();
        do
        {
                printf("\n1. Insert\n2. Delete\n3. Display\n4. Exit");
                printf("\nEnter the choice: ");
                scanf("%d", &ch);
                flushall();
                switch(ch)
                        {

                                case 1: printf("\nEnter the character / item to be inserted: ");
                                        scanf("%c", &item);
                                        insert(item);
                                        break;
                                case 2: del();
                                        break;
                                case 3: display();
                                        break;
                                case 4: exit(0);
                                        break;
                        }
        }while(ch!=4);
        getch();
}
```

SAMPLE POUTPUT:

```
1. Insert       2. Delete       3. Display      4. Exit
Enter the choice: 1
Enter the character / item to be inserted: A

1. Insert       2. Delete       3. Display      4. Exit
Enter the choice: 1
Enter the character / item to be inserted: B

1. Insert       2. Delete       3. Display      4. Exit
```

```
Enter the choice: 1
Enter the character / item to be inserted: C

1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 1
Enter the character / item to be inserted: D

1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 3

Contents of Queue is:
A B C D

1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 1
Enter the character / item to be inserted: F

Queue is Full

1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 2

Deleted item is: A

1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 2

Deleted item is: B

1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 3

Contents of Queue is:
C D

1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 1
Enter the character / item to be inserted: K

1. Insert 2. Delete 3. Display 4. Exit
the choice: 3

Contents of Queue is:
C D K
1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 4
```

EXPERIMENT - 07

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

a. Create a **SLL** of **N** Students Data by using ***front insertion***.
b. Display the status of **SLL** and count the number of nodes in it
c. Perform Insertion and Deletion at End of **SLL**
d. Perform Insertion and Deletion at Front of **SLL**
e. Demonstrate how this **SLL** can be used as **STACK** and **QUEUE**
f. Exit

ABOUT THE EXPERIMENT:

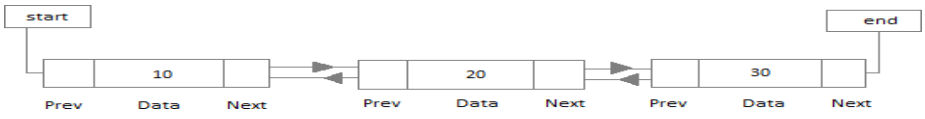
Linked List is a linear data structure and it is very common data structure which consists of group of nodes in a sequence which is divided in two parts. Each node consists of its own data and the address of the next node and forms a chain. Linked Lists are used to create trees and graphs.

- They are a dynamic in nature which allocates the memory when required.
- Insertion and deletion operations can be easily implemented.
- Stacks and queues can be easily executed.
- Linked List reduces the access time.
- Linked lists are used to implement stacks, queues, graphs, etc.
- Linked lists let you insert elements at the beginning and end of the list.
- In Linked Lists we don't need to know the size in advance.

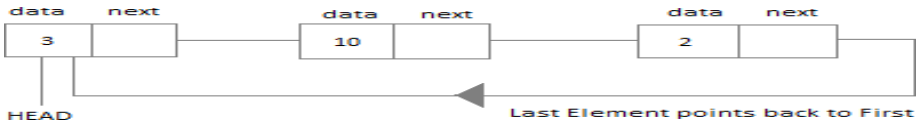
Types of Linked List:

Singly Linked List: Singly linked lists contain nodes which have a data part as well as an address part i.e. next,which points to the next node in sequence of nodes. The operations we can perform on singly linked lists are insertion, deletion and traversal.

Doubly Linked List: In a doubly linked list, each node contains two links the first link points to the previous node and the next link points to the next node in the sequence.



Circular Linked List: In the circular linked list the last node of the list contains the address of the first node and forms a circular chain.



ALGORITHM:

Step 1: Start.
Step 2: Read the value of N. (N student's information)
Step 3: Create a singly linked list. (SLL)
Step 4: Display the status of SLL.
Step 5: Count the number of nodes.
Step 6: Perform insertion at front of list.
Step 7: Perform deletion at the front of the list.
Step 8: Perform insertion at end of the list.
Step 9: Perform deletion at the end of the list.
Step 10: Demonstrate how singly linked list can be used as stack.
Step 11: Demonstrate how singly linked list can be used as queue.
Step 12: Stop.

PROGRAM CODE:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#define null 0 struct
student
{
      char usn[15],name[20],branch[10];
       int sem;
      char phno[20];
      struct student *link;
 };
typedef struct student node;
node *start;

void main()
{
void create(),insert_end(),del_front(),disp();
int ch;
clrscr();
while(1)
      {
                printf("Main Menu\n");
                printf("1:Create\n2:Display\n3:Insert Endt\n4:Delete Front\n5:Exit\n");
                 printf("Enter your choice\n");
                scanf("%d",&ch);
                switch(ch)
                   {
                        case 1:create();
                                break;
                        case 2:disp();
                                break;
                        case 3:insert_end();
                                break;
                        case 4:del_front();
                                break;
                        case 5:exit(0);
                }
```

```
        }
  }

void create()
        {
                int i,n; node *p;
                printf("Enter the number of students\n");
                scanf("%d",&n);
                for(i=0;i<n;i++)
                {
                        p=(node*)malloc(sizeof(node));
                        printf("Enter the student USN , NAME ,BRANCH,SEM,PHNO\n");
                        scanf("%s%s%s%d%s",
                        p->usn,p->name,p->branch,&p->sem,p->phno);
                        p->link=start;
                        start=p;
                }
        }

 void disp()
{
        int cnt=0;
        node *t;
        t=start;
        while(t)
   {
        cnt++;
        printf("%s\t%s\t%s\t%d\t%s->\n",t->usn,t->name,t->branch,t->sem,t->phno);
        t=t->link;
    }
        printf("Total number of nodes=%d\n",cnt);
 }

void insert_end()
        {
                node *p,*r; p=(node*)malloc(sizeof(node));
                printf("Enter the student USN , NAME ,BRANCH,SEM,PHNO\n");
                scanf("%s%s%s%d%s",p->usn,p->name,p->branch,&p->sem,p->phno);
                r=start;
                while(r->link!=null)
                   r=r->link;
                   r->link=p;
                   p->link=null;
        }

  void del_front()
  {
        node *q;
         if(start==null)
        {
                printf("List empty\n");
                return;
        }
        q=start;
```

```
    printf("Deleted node is %s",q->usn);
    start=start->link;
    free(q);
}
```

SAMPLE OUTPUT:

```
1. Create       2. Display      3. Insert End    4. Delete Front 5. Exit
Enter your choice: 1
Enter the number of students: 2
Enter the student USN, Name, Branch, Sem, Ph.No
1ep12cs001      kumar           cs      3       9900099000

Enter USN, Name, Branch, Sem, Ph.No
1ep12is002      ravi            is      3       9900099111

1. Create       2. Display      3. Insert       4. Delete       5. Exit
Enter your choice: 2

1ep12is002      ravi            is              3       9900099111
1ep12cs001      kumar           cs              3       9900099000
Total number of nodes:          2

1. Create   2. Display      3. Insert end    4. Delete front  5. Exit

Enter your choice: 3
Enter the student USN , NAME ,BRANCH,SEM,PHNO
1ep12cs003      suresh          cs      3       99000992222

1. Create       2. Display      3. Insert End    4. Delete Front         5. Exit
Enter your choice: 2

1ep12is002      ravi            is              3       9900099111
1ep12cs001      kumar           cs              3       9900099000
1ep12cs003      suresh          cs              3       99000992222

Total number of nodes:          3


1. Create       2. Display      3. Insert End    4. Delete Front         5. Exit

Enter your choice :4
The node deleted is 1ep12is002

1. Create       2. Display      3. Insert End    4. Delete Front         5. Exit

Enter your choice :2

1ep12cs001      kumar           cs              3       9900099000
1ep12cs003      suresh          cs              3       99000992222
```

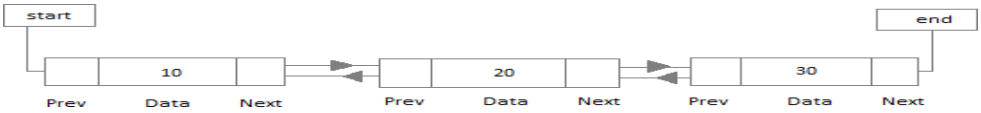
EXPERIMENT - 08

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

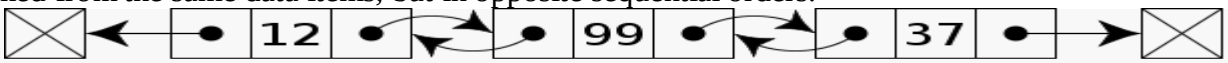
a. Create a **DLL** of **N** Employees Data by using ***end insertion***.
b. Display the status of **DLL** and count the number of nodes in it
c. Perform Insertion and Deletion at End of **DLL**
d. Perform Insertion and Deletion at Front of **DLL**
e. Demonstrate how this **DLL** can be used as **Double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

ABOUT THE EXPERIMENT:

Doubly Linked List: In a doubly linked list, each node contains two links the first link points to the previous node and the next link points to the next node in the sequence.



In computer science, a **doubly linked list** is a linked data structure that consists of a set of sequentially linked records called nodes. Each node contains two fields, called *links*, that are references to the previous andto the next node in the sequence of nodes. The beginning and ending nodes' **previous** and **next** links, respectively, point to some kind of terminator, typically a sentinel node or null, to facilitate traversal of the list. If there is only one sentinel node, then the list is circularly linked via the sentinel node. It can be conceptualized as two singly linked lists formed from the same data items, but in opposite sequential orders.



A doubly linked list whose nodes contain three fields: an integer value, the link to the next node, and the link to the previous node.

The two-node links allow traversal of the list in either direction. While adding or removing a node in a doubly linked list requires changing more links than the same operations on a singly linked list, the operations are simpler and potentially more efficient (for nodes other than first nodes) because there is no need to keep track ofthe previous node during traversal or no need to traverse the list to find the previous node, so that its link can be modified.

ALGORITHM:

Step 1: Start.
Step 2: Read the value of N. (N student's information)
Step 3: Create a doubly linked list. (DLL)
Step 4: Display the status of DLL.
Step 5: Count the number of nodes.
Step 6: Perform insertion at front of list.
Step 7: Perform deletion at the front of the list.
Step 8: Perform insertion at end of the list.
Step 9: Perform deletion at the end of the list.
Step 10: Demonstrate how doubly linked list can be used as double ended queue.
Step 11: Stop.

Program Code

```
#include<stdio.h>
#include<stdlib.h>
#define null 0
struct emp
{
   char name[40],dept[40],desig[40];
   int ssn;
   long int sal;
   char phno[20];
   struct emp *llink;
   struct emp *rlink;
};
typedef struct emp node;
node *start;
void create(),insert_front(),del_front(),disp();
void main()
{
int ch;
clrscr();
while(1)
  {
      printf("\nMain Menu\n");
      printf("1:Create\n2:Display\n3:Insert_Front\n4:Del_Front\n5:Exit\n");
      printf("Enter your choice\n");
      scanf("%d",&ch);
      switch(ch)
         {
               case 1:create();
                      break;
                case 2:disp();
                      break;
               case 3:insert_front();
                      break;
               case 4:del_front();
                      break;
               case 5:exit(0);
          }
    }
}
void create()
 {
      node *p, *t;
      int i, n;
      printf("Enter the number of employees \n");
      scanf("%d", &n);
      printf("Enter the employee details[SSN,NAME,DEPT,DESIG,SAL AND PH.NO.]\n");
      for(i=0; i<n; i++)
         {
               printf("enter the Employee %d details\n",i+1);
               p=(node*)malloc(sizeof(node));
               p->rlink=null;
```

```
            scanf("%d%s%s%s%ld%s",&p->ssn,p->name,p->dept,p->desig,&p->sal,p->phno);
            if(start==null)
                {
                    start=p;
                    start->llink=null;
                 }
             else
                    {
                        t=start;
                        while(t->rlink!=null)
                          t=t->rlink;
                          t->rlink=p;
                          p->llink=t;
                    }
          }
  }
}
 void disp()
 {
      node *r;
      r=start;
      while(r)
      {
            printf("|%d|%s|%s|%s|%ld|%s|\n<->",r->ssn,r->name,r->dept,r->desig,r->sal,r->phno);
            r=r->rlink;
      }
 }
 void insert_front()
 {
      node *p; p=(node*)malloc(sizeof(node));
      printf("Enter emp details\n");
      scanf("%d%s%s%s%ld%%s", &p->ssn, p->name,p->dept, p->desig, &p->sal, p->phno);
      p->rlink=start;
      start=p;
      start->llink=null;
 }
 void del_front()
 {
      node *q;
      if(start==null)
      {
            printf("list empty\n");
            return;
      }
      q=start;
      printf("Deleted node is %d",q->ssn);
      start=start->rlink;
      free(q);
      }
```

SAMPLE OUTPUT:

```
----------Employee Database----------
1. Create     2. Display     3. Insert front 4. Delete front 5. Exit
Enter your choice: 1

How many employees data you want to create:          2
Enter SSN, Name, Dept, Designation, Sal, Ph.No
1     KUMAR     CSE          INSTRUCTOR     8000          900099000
Enter SSN, Name, Dept, Designation, Sal, Ph.No
2     RAVI  ISE          INSTRUCTOR     9000          900099111

1. Create     2. Display     3. Insert front  4. Delete front          5.Exit
Enter your choice: 2

1     KUMAR     CSE          INSTRUCTOR     8000          900099000
2     RAVI      ISE          INSTRUCTOR     9000          900099111

The no. of nodes in list is:     2

1. Create     2. Display     3. Insert front  4. Delete front          5. Exit
Enter your choice: 3

   Enter SSN, Name, Dept, Designation, Sal, Ph.No
3     JIM     CSE          ATTENDER       6000          900099333

1. Create     2. Display     3. Insert      4. Delete      5. Exit
Enter your choice: 2
3     JIM       CSE          ATTENDER       6000          900099333
1     KUMAR     CSE          INSTRUCTOR     8000          900099000
2     RAVI      ISE          INSTRUCTOR     9000          900099111

The no. of nodes in list is:     3

1. Create     2. Display     3. Insert front  4. Delete front          5. Exit
Enter your choice: 4
The deleted node is  3

1. Create     2. Display     3. Insert front  4. Delete front          5.Exit
Enter your choice: 5
```

EXPERIMENT - 09

Design, Develop and Implement a Program in C for the following operations on **Singly Circular Linked List (SCLL)** with header nodes

a. Represent and Evaluate a Polynomial **P(x,y,z) = 6x2y2z-4yz5+3x3yz+2xy5z-2xyz3**

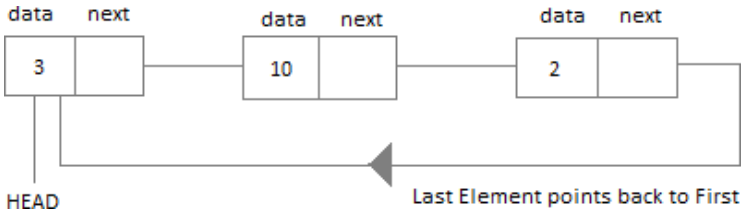
b. Find the sum of two polynomials **POLY1(x,y,z)** and **POLY2(x,y,z)** and store the result in POLYSUM(x,y,z)

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

ABOUT THE EXPERIMENT:

Circular Linked List:

In the circular linked list the last node of the list contains the address of the first node and forms a circular chain.

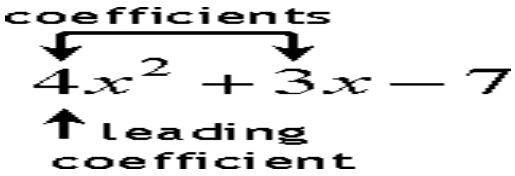


Polynomial:

A **polynomial equation** is an **equation** that can be written in the form. $ax^n + bx^{n-1} + \ldots + rx + s = 0$, where a, b, . . . , r and s are constants. We call the largest exponent of x appearing in a non-zero term of a **polynomial** the degree of that **polynomial**.

As with **polynomials** with one variable, you must pay attention to the rules of exponents and the order of operations so that you correctly **evaluate** an expression with two or more variables. **Evaluate** $x^2 + 3y^3$ for x = 7 and y = −2. Substitute the given values for x and y. **Evaluate** $4x^2y – 2xy^2 + x – 7$ for x = 3 and y = −1.

When a term contains both a number and a variable part, the number part is called the "coefficient". The coefficient on the leading term is called the "leading" coefficient.



In the above example, the coefficient of the leading term is 4; the coefficient of the second term is 3; the constant term doesn't have a coefficient.

Here are the steps required for Evaluating Polynomial Functions:

Step 1: Replace each x in the expression with the given value.

Step 2: Use the order of operation to simplify the expression.

Example 1 – Given $f(x) = -2x^2 + 5x - 7$, find $f(3)$.

Step 1: Replace each x in the expression with the given value. In this case, we replace each x with 3.

$$f(3) = -2(3)^2 + 5(3) - 7$$

Step 2: Use the order of operation to simplify theexpression.

$$f(3) = -2(9) + 5(3) - 7$$
$$f(3) = -18 + 15 - 7$$
$$f(3) = -10$$

Here are the steps required for addition of two polynomials.

Step 1

- Arrange the Polynomial in standard form
- Standard form of a polynomial just means that the term with highest degree is first and each of the following terms

Step 2

- Arrange the like terms in columns and add the like terms

Example 1: *Let's find the sum of the following two polynomials*
$(3y^5 - 2y + y^4 + 2y^3 + 5)$ and $(2y^5 + 3y^3 + 2+$

1) Write in Standard form $(3y^5 + y^4 + 2y^3 - 2y + 5) + (2y^5 + 3y^3 + 7y + 2)$

2) Arrange in columns of like terms and then add

$$\begin{array}{r} 3y^5 + y^4 + 2y^3 - 2y + 5 \\ 2y^5 \quad\quad + 3y^3 + 7y + 2 \\ \hline 5y^5 + y^4 + 5y^3 + 5y + 7 \end{array}$$

© mathwarehouse.com

ALGORITHM:

Step 1: Start.
Step 2: Read a polynomial.
Step 3: Represent the polynomial using singly circular linked list.
Step 3: Evaluate the given polynomial
Step 4: Read two polynomials and find the sum of the polynomials.
Step 5: Stop

PROGRAM CODE:

```
#include<stdio.h>
#include<alloc.h>
#include<math.h>
struct node
{
        int cf, px, py, pz;
        int flag;
        struct node *link;
};
```

```
typedef struct node NODE;

NODE* getnode()
{
	NODE *x;
	x=(NODE*)malloc(sizeof(NODE));
	if(x==NULL)
	{
		printf("Insufficient memory\n");exit(0);
	}
	return x;
}


void display(NODE *head)
{
	NODE *temp;
	if(head->link==head)
	{
		printf("Polynomial does not exist\n");return;
	}
	temp=head->link;
	printf("\n");
	while(temp!=head)
	{
		printf("%d x^%d y^%d z^%d",temp->cf,temp->px,temp->py,temp->pz);
		if(temp->link != head)
		printf(" + ");
		temp=temp->link;
	}
	printf("\n");
}


NODE* insert_rear(int cf,int x,int y,int z,NODE *head)
  {
	NODE *temp,*cur;
	temp=getnode();
	temp->cf=cf;
	temp->px=x;
	temp->py=y;
	temp->pz=z;
	cur=head->link;
	while(cur->link!=head)
	{
		cur=cur->link;
	}
	cur->link=temp;
	temp->link=head;return head;
 }
```

```
NODE* read_poly(NODE *head)
{
        int px, py, pz, cf, ch;
        printf("\nEnter coeff: ");
        scanf("%d",&cf);
        printf("\nEnter x, y, z powers(0-indiacate NO term): ");
        scanf("%d%d%d", &px, &py, &pz);
        head=insert_rear(cf,px,py,pz,head);
        printf("\nIf you wish to continue press 1 otherwise 0: ");
        scanf("%d", &ch);
        while(ch != 0)
        {
                printf("\nEnter coeff: ");
                scanf("%d",&cf);
                printf("\nEnter x, y, z powers(0-indiacate NO term): ");
                scanf("%d%d%d", &px, &py, &pz);
                head=insert_rear(cf,px,py,pz,head);
                printf("\nIf you wish to continue press 1 otherwise 0: ");
                scanf("%d", &ch);
        }
        return head;
}

NODE* add_poly(NODE *h1,NODE *h2,NODE *h3)
{
        NODE *p1,*p2;
        int x1,x2,y1,y2,z1,z2,cf1,cf2,cf;
        p1=h1->link;
        while(p1!=h1)
        {
                x1=p1->px;
                y1=p1->py;
                z1=p1->pz;
                cf1=p1->cf;
                p2=h2->link;
                while(p2!=h2)
                {
                        x2=p2->px;
                        y2=p2->py;
                        z2=p2->pz;
                        cf2=p2->cf;
                        if(x1==x2 && y1==y2 && z1==z2)
                        break;
                        p2=p2->link;
                }
                if(p2!=h2)
                {
                    cf=cf1+cf2;
                    p2->flag=1;
                    if(cf!=0)
                    h3=insert_rear(cf,x1,y1,z1,h3);
                else
```

```
            h3=insert_rear(cf1,x1,y1,z1,h3)
            p1=p1->link;

            p2=h2->link;
            while(p2!=h2)
            {
               if(p2->flag==0)
               h3=insert_rear(p2->cf,p2->px,p2->py,p2->pz,h3);
               p2=p2->link;
             }
        return h3;
 }

    void evaluate(NODE *h1)
    {
            NODE *head; int x, y, z;
            float result=0.0; head=h1;
            printf("\nEnter x, y, z, terms to evaluate:\n");
            scanf("%d%d%d", &x, &y, &z);
            while(h1->link != head)
                    {
                            result = result + (h1->cf * pow(x,h1->px) * pow(y,h1->py) * pow(z,h1-pz);
                            h1=h1->link;
                    }
            result = result + (h1->cf * pow(x,h1->px) * pow(y,h1->py) * pow(z,h1->pz));
            printf("\nPolynomial result is: %f", result);
    }

void main()
{
    NODE *h1,*h2,*h3;
     int ch;
     clrscr();
     h1=getnode();
     h2=getnode();
     h3=getnode();
     h1->link=h1;
     h2->link=h2;
     h3->link=h3;
    while(1)
      {
            printf("\n\n1.Evaluate polynomial\n2.Add two polynomials\n3.Exit\n");
            printf("Enter your choice: ");
            scanf("%d", &ch);
            switch(ch)
              {
                    case 1:
                            printf("\nEnter polynomial to evaluate:\n");
                            h1=read_poly(h1);
                            display(h1);
                            evaluate(h1);
                        break;
                    case 2:
                            printf("\nEnter the first polynomial:");
```

```
                        h1=read_poly(h1);
                        printf("\nEnter the second polynomial:");
                        h2=read_poly(h2);
                        h3=add_poly(h1,h2,h3);
                        printf("\nFirst polynomial is: ");
                        display(h1);
                        printf("\nSecond polynomial is: ");
                        display(h2);
                        printf("\nThe sum of 2 polynomials is: ");
                        display(h3);
                        break;
                case 3: exit(0);
                        break;
                default:printf("\nInvalid entry");
                        break;
        }
        getch();
    }
}
```

SAMPLE OUTPUT:

```
1.      Evaluate polynomial P(x,y,z) = 6x2y2z-4yz5+3x3yz+2xy5z-2xyz3
2.      Add two polynomials
3.      Exit
Enter your choice:      1
Enter polynomial to evaluate:
Enter coeff:    6
Enter x, y, z powers (0-indiacate NO term: 2
If you wish to continue press 1 otherwise 0: 1    2       1
Enter coeff: -4
Enter x, y, z powers (0-indiacate NO term: 0      1       5
If you wish to continue press 1 otherwise 0:   1
Enter coeff: 3
Enter x, y, z powers (0-indiacate NO term: 3      1       1
If you wish to continue press 1 otherwise 0: 1
Enter coeff: 2
Enter x, y, z powers (0-indiacate NO term: 1      5       1
If you wish to continue press 1 otherwise 0:   1
Enter coeff: -2
Enter x, y, z powers (0-indiacate NO term: 1
1
3
If you wish to continue press 1 otherwise 0: 0
```

Polynomial is:

6 x^2 y^2 z^1 + -4 x^0 y^1 z^5 + 3 x^3 y^1 z^1 + 2 x^1 y^5 z^1 + -2 x^1 y^1 z^3

Enter x, y, z, terms to evaluate:

1 1 1

Polynomial result is: 5.000000

1. Evaluate polynomial P(x,y,z) = 6x2y2z-4yz5+3x3yz+2xy5z-2xyz3
2. Add two polynomials
3. Exit
Enter your choice:

2Enter 1st

polynomial:

Enter coeff: 4
Enter x, y, z powers (0-indiacate NO term: 22 2

If you wish to continue press 1 otherwise
0: 1Enter coeff: 3
Enter x, y, z powers (0-indiacate NO term: 11

2If you wish to continue press 1 otherwise 0:

0 Enter 2nd polynomial:

Enter coeff: 6
Enter x, y, z powers (0-indiacate NO term: 22
2If you wish to continue press 1 otherwise 0: 0

Polynomial is:
1st Polynomial is:
4 x^2 y^2 z^2 + 3 x^1 y^1 z^2

2nd Polynomial is:
6 x^2 y^2 z^2

Added polynomial: 10 x^2 y^2 z^2 + 3 x^1 y^1 z^2
1. Evaluate polynomial P(x,y,z) = 6x2y2z-4yz5+3x3yz+2xy5z-2xyz3
2. Add two polynomials
3. Exit
Enter your choice: 3

EXPERIMENT 10

Design, Develop and Implement a menu driven Program in C for the following operations on **Binary Search Tree (BST)** of Integers

a. Create a BST of **N** Integers: **6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2**
b. Traverse the BST in Inorder, Preorder and Post Order
c. Search the BST for a given element (**KEY**) and report the appropriate message
d. Delete an element (**ELEM**) from BST
e. Exit

ABOUT THE EXPERIMENT:

A **binary search tree** (BST) is a **tree** in which all nodes follows the below mentioned properties

- The left sub-**tree** of a node has key less than or equal to its parent node's key.
- The right sub-**tree** of a node has key greater than or equal to its parent node's key.

Thus, a binary search tree (BST) divides all its sub-trees into two segments; *left* sub-tree and *right* sub-tree and can be defined as

```
left_subtree (keys) ≤ node (key) ≤ right_subtree (keys)
```

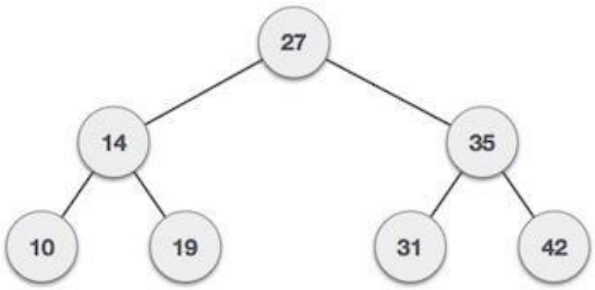


Fig: An example of BST

Following are basic primary operations of a tree which are following.

- **Search** − search an element in a tree.
- **Insert** − insert an element in a tree.
- **Preorder Traversal** − traverse a tree in a preorder manner.
- **Inorder Traversal** − traverse a tree in an inorder manner.
- **Postorder Traversal** − traverse a tree in a postorder manner.

Node definition: Define a node having some data, references to its left and right child nodes.

```
struct node {
   int data;
   struct node *leftChild;
   struct node *rightChild;
};
```

ALGORITHM:

Step 1: Start.
Step 2: Create a Binary Search Tree for N elements.Step
3: Traverse the tree in inorder.
Step 4: Traverse the tree in preorder Step
6: Traverse the tree in postorder.
Step 7: Search the given key element in the BST.Step
8: Delete an element from BST.
Step 9: Stop

PROGRAM CODE:

```
#include <stdio.h>
#include <stdlib.h>
struct BST
{
        int data;
        struct BST *left;
        struct BST *right;
};

typedef struct BST NODE;

NODE *node;

NODE* createtree(NODE *node, int data)
{
        if (node == NULL)
        {
                NODE *temp;
                temp= (NODE*)malloc(sizeof(NODE));
                temp->data = data;
                temp->left = temp->right = NULL;
                return temp;
        }

        if (data < (node->data))
        {
                node->left = createtree(node->left, data);
        }
        else if (data > node->data)
        {
                node -> right = createtree(node->right, data);
        }
        return node;
}
```

```
NODE* search(NODE *node, int data)
{
      if(node == NULL)
          printf("\nElement not found");
      else if(data < node->data)
       {
          node->left=search(node->left, data);
       }
       else if(data > node->data)
      {
          node->right=search(node->right, data);
       }
     else
           printf("\nElement found is: %d", node->data);
      return node;
}
 void inorder(NODE *node)
      {
                if(node != NULL)
                {
                      inorder(node->left);
                      printf("%d\t", node->data);
                       inorder(node->right);
                }
      }
 void preorder(NODE *node)
      {
                if(node != NULL)
                {
                      printf("%d\t", node->data);
                      preorder(node->left);
                      preorder(node->right);
                }
      }

      void postorder(NODE *node)
      {
                if(node != NULL)
                {
                          postorder(node->left);
                          postorder(node->right);
                          printf("%d\t", node->data);
                }
      }

 void main()
      {
         int data, ch, i, n;
         NODE *root=NULL;
         clrscr();
         while (1)
           {
               printf("\n1.Insertion in Binary Search Tree");
```

```
                printf("\n2.Inorder\n3.Preorder\n4.Postorder\n5.Exit");
                printf("\nEnter your choice: ");
                scanf("%d", &ch);
                switch (ch)
                 {
                        case 1: printf("\nEnter N value: " );
                                scanf("%d", &n);
                                printf("\nEnter the values to create BST like(6,9,5,2,8,15,24,14,7,8,5,2)\n");
                                for(i=0; i<n; i++)
                                {
                                        scanf("%d", &data);
                                        root=createtree(root, data);
                                }
                                break;
                        case 2: printf("\nInorder Traversal: \n");
                                inorder(root);
                                break;
                        case 3: printf("\nPreorder Traversal: \n");
                                preorder(root);
                                break;
                        case 4: printf("\nPostorder Traversal: \n");
                                postorder(root);
                                break;
                        case 5: exit(0);
                        default:printf("\nWrong option");
                                break;
                }
        }
}
```

SAMPLE OUTPUT:

1. Insertion in Binary Search Tree
2. Search Element in Binary Search Tree
3. Delete Element in Binary Search Tree
4. Inorder
5. Preorder
6. Postorder
7. Exit

Enter your choice: 1
Enter N value: 12
Enter the values to create BST like(6,9,5,2,8,15,24,14,7,8,5,2)
6 9 5 2 8 15 24 14 7 8 5 2

Insertion in Binary Search Tree
1. Search Element in Binary Search Tree
2. Delete Element in Binary Search Tree
3. Inorder
4. Preorder
5. Postorder
6. Exit

Enter your choice: 4

Inorder Traversal:
2 5 6 7 8 9 14 15 24
1. Insertion in Binary Search Tree
2. Search Element in Binary Search Tree
3. Delete Element in Binary Search Tree
4. Inorder
5. Preorder
6. Postorder
7. Exit
Enter your choice: 5

Preorder Traversal:
6 5 2 9 8 7 15 14 24

1. Insertion in Binary Search Tree
2. Search Element in Binary Search Tree
3. Delete Element in Binary Search Tree
4. Inorder
5. Preorder
6. Postorder
7. Exit
Enter your choice: 6

Postorder Traversal:
2 5 7 8 14 24 15 9 6

1. Insertion in Binary Search Tree
2. Search Element in Binary Search Tree
3. Delete Element in Binary Search Tree
Enter your choice: 2
Enter the element to search: 24
4. Inorder
5. Preorder
6. Postorder
7. Exit
Element found is: 24

1. Insertion in Binary Search Tree
2. Search Element in Binary Search Tree
3. Delete Element in Binary Search Tree
4. Inorder
5. Preorder
6. Postorder
7. Exit
Enter your choice: 2
Enter the element to search: 50

Element not found
1. Insertion in Binary Search Tree
2. Search Element in Binary Search Tree
3. Delete Element in Binary Search Tree
4. Inorder
5. Preorder
6. Postorder
7. Exit

Enter your choice: 3
Enter the element to search: 15

1. Insertion in Binary Search Tree
2. Search Element in Binary Search Tree
3. Delete Element in Binary Search Tree
4. Inorder
5. Preorder
6. Postorder
7. Exit
Enter your choice: 4

Inorder Traversal:
2 5 6 7 8 9 14 24

1. Insertion in Binary Search Tree
2. Search Element in Binary Search Tree
3. Delete Element in Binary Search Tree
4. Inorder
5. Preorder
6. Postorder
7. Exit
Enter your choice: 5
Preorder Traversal:
6 5 2 9 8 7 24 14

1. Insertion in Binary Search Tree
2. Search Element in Binary Search Tree
3. Delete Element in Binary Search Tree
4. Inorder
5. Preorder
6. Postorder
7. Exit
Enter your choice: 7

EXPERIMENT 11

Design, Develop and Implement a Program in C for the following operations on **Graph(G)** of Cities

a. Create a Graph of **N** cities using Adjacency Matrix.
b. Print all the nodes **reachable** from a given starting node in a digraph using **BFS** method
c. Check whether a given graph is **connected** or not using **DFS** method.

ABOUT THE EXPERIMENT:

Adjacency Matrix

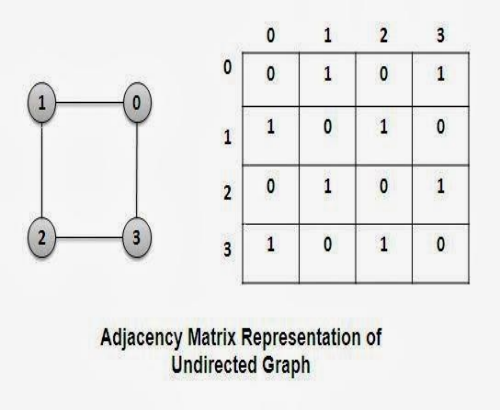
In **graph** theory, computer science, an **adjacency matrix** is a square **matrix** used to **represent** a finite **graph**. The elements of the **matrix** indicate whether pairs of vertices are adjacent or not in the **graph**. In the special case of a finite simple **graph**, the **adjacency matrix** is a (0, 1)-**matrix** with zeros on its diagonal.

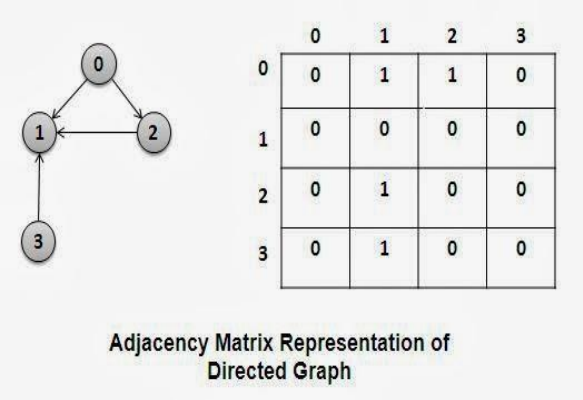
A graph G = (V, E) where v= {0, 1, 2, . . .n-1} can be represented using two dimensional integer array of size n x n.

a[20][20] can be used to store a graph with 20 vertices.
a[i][j] = 1, indicates presence of edge between two vertices i and j.
a[i][j] = 0, indicates absence of edge between two vertices i and j.

- A graph is represented using square matrix.
- Adjacency matrix of an undirected graph is always a symmetric matrix, i.e. an edge (i, j) implies the edge (j, i).
- Adjacency matrix of a directed graph is never symmetric, adj[i][j] = 1 indicates a directed edge from vertex i to vertex j.

An example of adjacency matrix representation of an undirected and directed graph is given below:

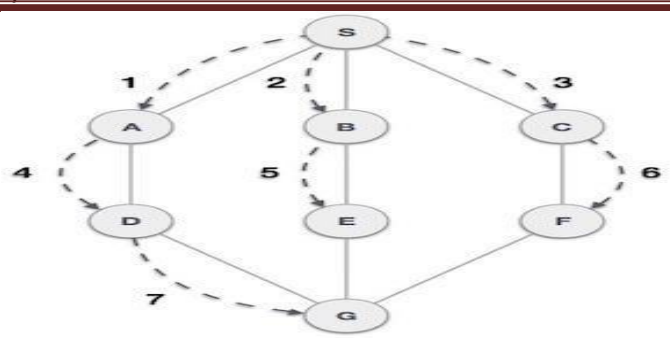




BFS

Breadth-first search (**BFS**) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root and explores the neighbor nodes first, before moving to the next level neighbors.

Breadth First Search algorithm(BFS) traverses a graph in a breadth wards motion and uses a queue to remember to get the next vertex to start a search when a dead end occurs in any iteration.



As in example given above, BFS algorithm traverses from A to B to E to F first then to C and G lastly to D. It employs following rules.

- **Rule 1** − Visit adjacent unvisited vertex. Mark it visited. Display it. Insert it in a queue.
- **Rule 2** − If no adjacent vertex found, remove the first vertex from queue.
- **Rule 3** − Repeat Rule 1 and Rule 2 until queue is empty.

DFS

Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. One starts at the root (selecting some arbitrary node as the root in the case of a graph) and explores as far as possible along each branch before backtracking.

Depth-first search, or **DFS**, is a way to traverse the graph. Initially it allows visiting vertices of the graph only, but there are hundreds of algorithms for graphs, which are based on DFS. Therefore, understanding the principles of depth-first search is quite important to move ahead into the graph theory. The principle of the algorithm is quite simple: to go forward (in depth) while there is such possibility, otherwise to backtrack.

Algorithm

In DFS, each vertex has three possible colors representing its state:

white: vertex is unvisited;

gray: vertex is in progress;

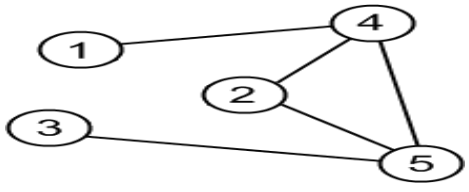
black: DFS has finished processing the vertex.

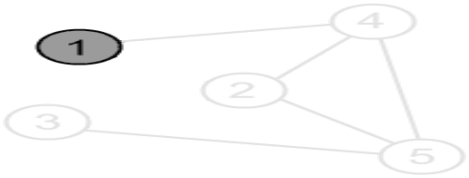
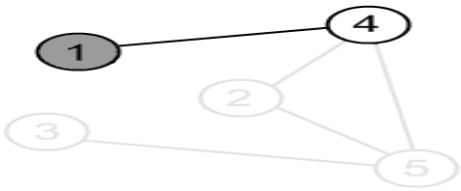
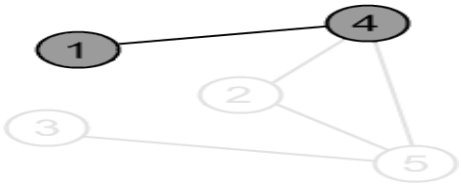
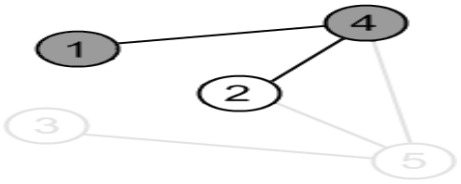
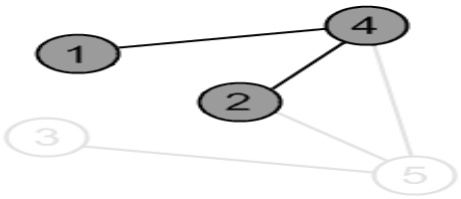
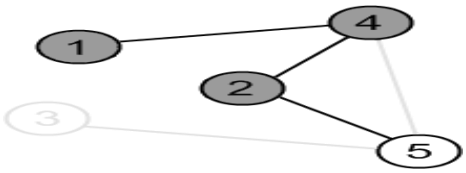
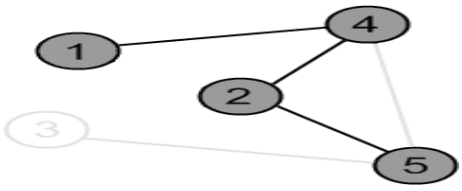
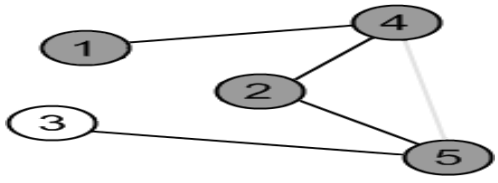
NB. For most algorithms Boolean classification *unvisited / visited* is quite enough, but we show general case here.

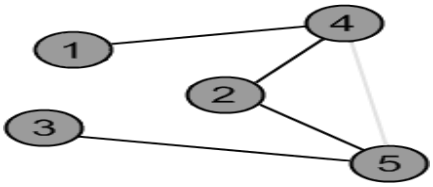
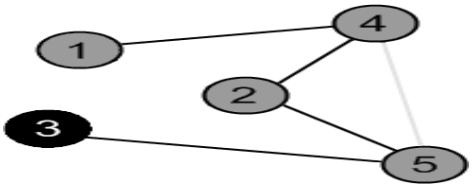
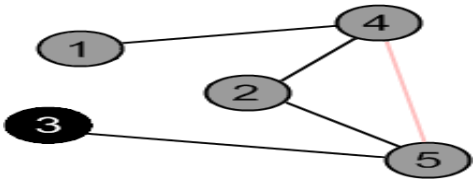
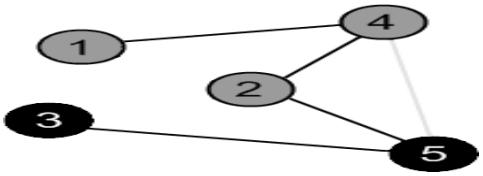
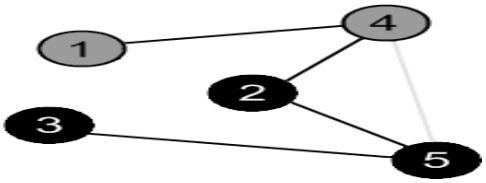
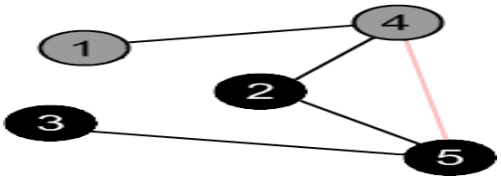
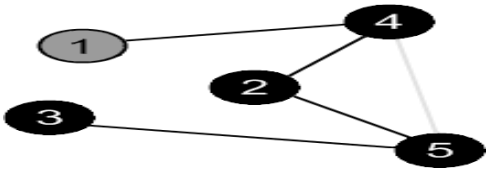
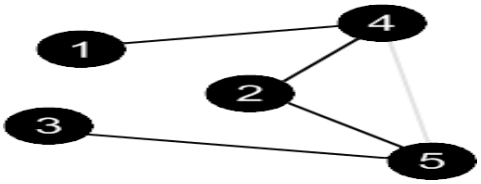
Initially all vertices are white (unvisited). DFS starts in arbitrary vertex and runs as follows:

- Mark vertex **u** as gray (visited).
- For each edge **(u, v)**, where **u** is white, run depth-first search for **u** recursively.
- Mark vertex **u** as black and backtrack to the parent.

Example. Traverse a graph shown below, using DFS. Start from a vertex with number 1.

	Source graph.

	Mark a vertex **1** as gray.
	There is an edge **(1, 4)** and a vertex **4** is unvisited. Go there.
	Mark the vertex **4** as gray.
	There is an edge **(4, 2)** and vertex a **2** is unvisited. Go there.
	Mark the vertex **2** as gray.
	There is an edge **(2, 5)** and a vertex **5** is unvisited. Go there.
	Mark the vertex **5** as gray.
	There is an edge **(5, 3)** and a vertex **3** is unvisited. Go there.

	Mark the vertex **3** as gray.
	There are no ways to go from the vertex **3**. Mark it as black and backtrack to the vertex **5**.
	There is an edge **(5, 4)**, but the vertex 4 is gray.
	There are no ways to go from the vertex **5.** Mark it as black and backtrack to the vertex **2**.
	There are no more edges, adjacent to vertex **2.** Mark it as black and backtrack to the vertex **4**.
	There is an edge **(4, 5)**, but the vertex 5 is black.
	There are no more edges, adjacent to the vertex **4.** Mark it as black and backtrack to the vertex **1**.
	There are no more edges, adjacent to the vertex **1.** Mark it as black. DFS is over.

ALGORITHM:

Step 1: Start.
Step 2: Input the value of N nodes of the graph
Step 3: Create a graph of N nodes using adjacency matrix representation.
Step 4: Print the nodes reachable from the starting node using BFS.
Step 5: Check whether graph is connected or not using DFS.
Step 6: Stop.

PROGRAM CODE:

```
#include<stdio.h>
#include<conio.h>
int a[10][10], n, m, i, j, source, s[10], b[10];
int visited[10];

void create()
{
        printf("\nEnter the number of vertices of the digraph: ");
        scanf("%d", &n);
        printf("\nEnter the adjacency matrix of the graph:\n");
        for(i=1; i<=n; i++)
        for(j=1; j<=n; j++)
        scanf("%d", &a[i][j]);
}

void bfs()
{
   int q[10], u, front=0, rear=-1;
   printf("\nEnter the source vertex to find other nodes reachable or not: ");
   scanf("%d", &source);
   q[++rear] = source;
   visited[source] = 1;
   printf("\nThe reachable vertices are: ");
   while(front<=rear)
        {
            u = q[front++];
            for(i=1; i<=n; i++)
            {
                if(a[u][i] == 1 && visited[i] == 0)
                {
                        q[++rear] = i;
                        visited[i] = 1;
                        printf("\n%d", i);
                }
            }
      }
}

 void dfs(int source)
    {
       int v, top = -1;
```

```
   s[++top] = 1;
   b[source] = 1;

for(v=1; v<=n; v++)
        {
          if(a[source][v] == 1 && b[v] == 0)
                {
                        printf("\n%d -> %d", source, v);
                        dfs(v);
                }
        }
}

void main()
{
        int ch;
        clrscr();
        while(1)
        {
                printf("\n1.Create Graph\n2.BFS\n3.Check graph connected or not(DFS)\n4.Exit");printf("\nEnter your
                choice:  ");
                scanf("%d", &ch);
                switch(ch)
                {
                        case 1: create();
                                break;
                        case 2: bfs();
                                for(i=1;i<=n;i++)
                                if(visited[i]==0)
                                printf("\the vertex that is not reachable  %d" ,i);
                                break;
                        case 3: printf("\nEnter the source vertex to find the connectivity:  ");
                                scanf("%d", &source);
                                m=1;
                                dfs(source);
                                for(i=1;i<=n;i++)
                                {
                                        if(b[i]==0)
                                       m=0;
                                }
                                if(m==1)
                                printf("\n Graph is Connected");
                                else
                                printf("\n Graph is not Connected");
                                break;
                                default: exit(0);
                        }
                }
        }
```

SAMPLE OUTPUT:

```
1. Create Graph     2.BFS     3.Check graph connected or not (DFS)     4.Exit
Enter your choice: 1

Enter the number of vertices of the digraph: 4Enter the

adjacency matrix of the graph:
0    0    1    1
0    0    0    0
0    1    0    0
0    1    0    0
1. Create Graph     2.BFS     3.Check graph connected or not (DFS)     4.Exit
Enter your choice: 2

Enter the source vertex to find other nodes reachable or not: 13
4
2

1. Create Graph     2.BFS     3.Check graph connected or not (DFS)     4.Exit
Enter your choice: 3

Enter the source vertex to find the connectivity: 11 ->
3
3 -> 2
1 -> 4
```

Graph is Connected

```
1. Create Graph     2.BFS     3.Check graph connected or not (DFS)     4.Exit
Enter your choice: 4
```

EXPERIMENT - 12

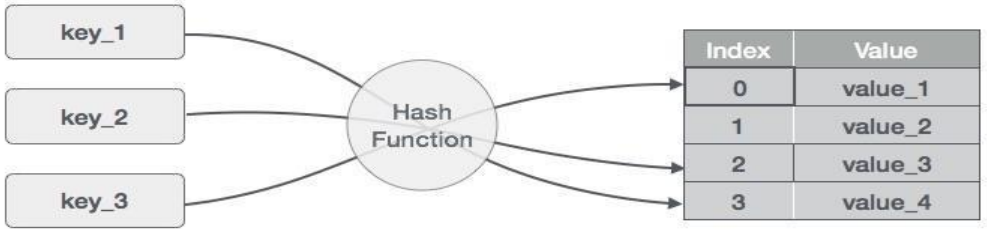
Given a File of **N** employee records with a set **K** of Keys(4-digit) which uniquely determine the records in file **F**. Assume that file **F** is maintained in memory by a Hash Table(HT) of **m** memory locations with **L** as the set of memory addresses (2-digit) of locations in HT. Let the keys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

ABOUT THE EXPERIMENT:

Hash Table is a data structure which store data in associative manner. In hash table, data isstored in array format where each data values has its own unique index value. Access of databecomes very fast if we know the index of desired data.

Thus, it becomes a data structure in which insertion and search operations are very fast irrespective of size of data. Hash Table uses array as a storage medium and uses hash technique to generate index where an element is to be inserted or to be located from.

Hashing: Hashing is a technique to convert a range of key values into a range of indexes of an array.We're going to use modulo operator to get a range of key values. Consider an example of hash tableof size 20, and following items are to be stored. Item are in (key, value) format.



(1,20) (2,70) (42,80) (4,25) (12,44) (14,32) (17,11) (13,78) (37,98)

S.n.	Key	Hash	Array Index
1	1	1 % 20 = 1	1
2	2	2 % 20 = 2	2
3	42	42 % 20 = 2	2
4	4	4 % 20 = 4	4
5	12	12 % 20 = 12	12
6	14	14 % 20 = 14	14

7	17	17 % 20 = 17	17
8	13	13 % 20 = 13	13
9	37	37 % 20 = 17	17

Linear Probing

As we can see, it may happen that the hashing technique used create already used index of thearray. In such case, we can search the next empty location in the array by looking into the nextcell until we found an empty cell. This technique is called linear probing.

S.n.	Key	Hash	Array Index	After Linear Probing, Array Index
1	1	1 % 20 = 1	1	1
2	2	2 % 20 = 2	2	2
3	42	42 % 20 = 2	2	3
4	4	4 % 20 = 4	4	4
5	12	12 % 20 = 12	12	12
6	14	14 % 20 = 14	14	14
7	17	17 % 20 = 17	17	17
8	13	13 % 20 = 13	13	13
9	37	37 % 20 = 17	17	18

BasicOperations

Following are basic primary operations of a hash table which are following.

- **Search** − search an element in a hash table.
- **Insert** − insert an element in a hash table.
- **delete** − delete an element from a hash table.

DataItem
Define a data item having some data, and key based on which search is to be conducted in hashtable.

ALGORITHM:
Step 1: Start.
Step 2: Given a File of N employee records with a set K of Keys (4-digit) which uniquely determine the records in file F.
Step 3: 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.
Step 3: Let the keys in K and addresses in L are Integers
Step 4: Hash function H: K ®L as H(K)=K mod m (remainder method)
Step 5: Hashing as to map a given key K to the address space L, Resolve the collision (if any) is using linear probing.
Step6: Stop.

PROGRAM CODE:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 10

struct employee
{
        int id;
        char name[15];
};
typedef struct employee EMP;
EMP emp[MAX];
int a[MAX];

int create(int num)
{
        int key;
        key = num % 100;
        return key;
}

int getemp(EMP emp[],int key)
{
        printf("\nEnter emp id: ");
        scanf("%d",&emp[key].id);
        printf("\nEnter emp name: ");
        flushall();
        gets(emp[key].name);
        return key;
}
void display()
{
        int i, ch;
        printf("\n1.Display ALL\n2.Filtered Display");
        printf("\nEnter the choice:                    ");
        scanf("%d",&ch);
        if(ch == 1)
```

```
        {
             printf("\nThe hash table is:\n");
             printf("\nHTKey\tEmpID\tEmpName");
             for(i=0; i<MAX; i++)
            printf("\n%d\t%d\t%s", i, emp[i].id, emp[i].name);
          }
        else
        {
                printf("\nThe hash table is:\n");
                printf("\nHTKey\tEmpID\tEmpName");
                for(i=0; i<MAX; i++)
                if(a[i] != -1)
                {
                        printf("\n%d\t%d\t%s", i, emp[i].id, emp[i].name);
                        continue;
                }
        }
}
void linear_prob(int key, int num)
{
        int flag, i, count = 0;flag = 0;
        if(a[key] == -1)
        {
                a[key]=getemp(emp, key);
        }
    else
        {       printf("\nCollision Detected...!!!\n");
                i = 0;
                while(i < MAX)
                {
                        if (a[i] != -1)
                        count++;
                        else
                        i++;
           printf("\nCollision avoided successfully using LINEAR PROBING\n");
           if(count == MAX)
          {
             printf("\n Hash table is full");
             display(emp);
             exit(1);
          }
          for(i=key; i<MAX; i++)

         if(a[i] == -1)
             {

                        a[i] = num;
                        flag = 1; break;
             }
             i = 0;
             while((i < key) && (flag == 0))
             {
```

```
                        if(a[i] == -1)
                        {
                            a[i] = num;
                            flag=1;
                            break;
                            } i++;
                            }
                    }
            }

  void main()
{
    int num, key, i;
    int ans = 1;
    clrscr();
    printf("\nCollision handling      by linear probing: ");
     for (i=0; i < MAX; i++)
    {
            a[i] = -1;
    }
    do
    {
            printf("\nEnter the data: ");
            scanf("%d", &num);
            key=create(num);
            linear_prob(key, num);
            printf("\nDo you wish to continue? (1/0): ");
            scanf("%d",&ans);
      }while(ans);
       display(emp);
       getch();
      }
```

SAMPLE OUTPUT:

```
RUN1:
Enter the data: 2
Enter emp id:      100
Enter emp name: Anand
Do you wish to continue? (1/0):     1
Enter the data: 4
Enter emp id:      101
Enter emp name: Kumar
Do you wish to continue? (1/0):     0

1.Display ALL
2.Filtered Display
Enter the choice: 1

The hash table is:
```

HTKey	EmpID	EmpName
0	0	
1	0	
2	100	Anand
3	0	
4	101	Kumar
5	0	
6	0	
7	0	
8	0	
9	0	

```
RUN2:
Enter the data: 2
Enter emp id:      100
Enter emp name: Anand
Do you wish to continue? (1/0):     1

Enter the data: 4
Enter emp id:      101
Enter emp name: Kumar
Do you wish to continue? (1/0):     0

1.Display ALL
2.Filtered Display
Enter the choice: 1

The hash table is:
```

HTKey	EmpID	EmpName
2	100	Anand
4	101	Kumar

VIVA QUESTIONS AND ANSWERS

Data structures refers to the way data is organized and manipulated. It seeks to find ways to make data access more efficient. When dealing with data structure, we not only focus on one piece of data, but rather different set of data and how they can relate to one another in an organized manner.

1) Differentiate file structure from storage structure.
Basically, the key difference is the memory area that is being accessed. When dealing with the structure that resides the main memory of the computer system, this is referred to as storage structure. When dealing with an auxiliary structure, we refer to it as file structures.

2) When is a binary search best applied?
A binary search is an algorithm that is best applied to search a list when the elements are already in order or sorted. The list is search starting in the middle, such that if that middle value is not the target search key, it will check to see if it will continue the search on the lower half of the list or the higher half. The split and search will then continue in the same manner.

3) What is a linked list?
A linked list is a sequence of nodes in which each node is connected to the node following it. This forms a chain-like link of data storage.

4) How do you reference all the elements in a one-dimension array?
To do this, an indexed loop is used, such that the counter runs from 0 to the array size minus one. In this manner, we are able to reference all the elements in sequence by using the loop counter as the array subscript.

5) In what areas do data structures applied?
Data structures is important in almost every aspect where data is involved. In general, algorithms that involve efficient data structure is applied in the following areas: numerical analysis, operating system, A.I., compiler design, database management, graphics, and statistical analysis, to name a few.

6) What is LIFO?
LIFO is short for Last In First Out, and refers to how data is accessed, stored and retrieved. Using this scheme, data that was stored last , should be the one to be extracted first. This also means that in order to gain access to the first data, all the other data that was stored before this first data must first be retrieved and extracted.

8) What is a queue?
A queue is a data structures that can simulates a list or stream of data. In this structure, new elements are inserted at one end and existing elements are removed from the other end.

9) What are binary trees?
A binary tree is one type of data structure that has two nodes, a left node and a right node. In programming, binary trees are actually an extension of the linked list structures.

10) Which data structures is applied when dealing with a recursive function?
Recursion, which is basically a function that calls itself based on a terminating condition, makes use of the stack. Using LIFO, a call to a recursive function saves the return address so that it knows how to return to the calling function after the call terminates.

11) What is a stack?
A stack is a data structure in which only the top element can be accessed. As data is stored in the stack, each data is pushed downward, leaving the most recently added data on top.

12) Explain Binary Search Tree
A binary search tree stores data in such a way that they can be retrieved very efficiently. The left subtree contains nodes whose keys are less than the node's key value, while the right subtree contains nodes whose keys are greater than or equal to the node's key value. Moreover, both subtrees are also binary search trees.

13) What are multidimensional arrays?
Multidimensional arrays make use of multiple indexes to store data. It is useful when storing data that cannot be represented using a single dimensional indexing, such as data representation in a board game, tables with data stored in more than one column.

14) Are linked lists considered linear or non-linear data structures?
It actually depends on where you intend to apply linked lists. If you based it on storage, a linked list is considered non-linear. On the other hand, if you based it on access strategies, then a linked list is considered linear.

15) How does dynamic memory allocation help in managing data?
Aside from being able to store simple structured data types, dynamic memory allocation can combine separately allocated structured blocks to form composite structures that expand and contract as needed.

16) What is FIFO?
FIFO is short for First-in, First-out, and is used to represent how data is accessed in a queue. Data has been inserted into the queue list the longest is the one that is removed first.

17) What is an ordered list?
An ordered list is a list in which each node's position in the list is determined by the value of its key component, so that the key valuesform an increasing sequence, as the list is traversed.

18) What is merge sort?
Merge sort takes a divide-and-conquer approach to sorting data. In a sequence of data, adjacent ones are merged and sorted to create bigger sorted lists. These sorted lists are then merged again to form an even bigger sorted list, which continuous until you have one single sorted list.

19) Differentiate NULL and VOID.
Null is actually a value, whereas Void is a data type identifier. A variable that is given a Null value simply indicates an empty value. Void is used to identify pointers as having no initial size.

20) What is the primary advantage of a linked list?
A linked list is a very ideal data structure because it can be modified easily. This means that modifying a linked list works regardlessof how many elements are in the list.

21) What is the difference between a PUSH and a POP?
Pushing and popping applies to the way data is stored and retrieved in a stack. A push denotes data being added to it, meaning data is being "pushed" into the stack. On the other hand, a pop denotes data retrieval, and in particular refers to the topmost data being accessed.

22) What is a linear search?
A linear search refers to the way a target key is being searched in a sequential data structure. Using this method, each element in the list is checked and compared against the target key, and is repeated until found or if the end of the list has been reached.

23) How does variable declaration affect memory allocation?
The amount of memory to be allocated or reserved would depend on the data type of the variable being declared. For example, if a variable is declared to be of integer type, then 32 bits of memory storage will be reserved for that variable.

24) What is the advantage of the heap over a stack?
Basically, the heap is more flexible than the stack. That's because memory space for the heap can be dynamically allocated and de- allocated as needed. However, memory of the heap can at times be slower when compared to that stack.

25) What is a postfix expression?
A postfix expression is an expression in which each operator follows its operands. The advantage of this form is that there is no needto group sub-expressions in parentheses or to consider operator precedence.

26).What is the difference between the HEAP and the STACK?
(Solution: HEAP is used to store dynamically allocated memory (malloc). STACK stores static data (int, const).)

27).Where in memory are HEAP and the STACK located relative to the executing program?
(Solution: The STACK and HEAP are stored "below" the executing program. The HEAP "grows" toward the program executable while the STACK grows away from it.)

28).Describe the data structures of a double-linked list.

(Solution: A double-linked list structure contains one pointer to the previous record in the list and a pointer to the next record in thelist plus the record data.)

29).How do you insert a record between two nodes in double-linked list?
(Solution: Previous R; Data R; Next R; To insert a record (B) between two others (A and C): Previous.B = A; Next.B = C; Next.A =B; Previous.C = B;)

30).In which data structure, elements can be added or removed at either end, but not in the middle?
(Solution: queue)

31).Which one is faster? A binary search of an orderd set of elements in an array or a sequential search of the elements.
(Solution: binary search)

32)What is a balanced tree?
(Solution: A binary tree is balanced if the depth of two subtrees of every node never differ by more than one)

34)Which data structure is needed to convert infix notations to post fix notations?
(Solution: stack)

35)What is data structure or how would you define data structure?
(Solution: In programming the term data structure refers to a scheme for organizing related piece of information. Data Structure = Organized Data + Allowed Operations.)

36).Which data structures we can implement using link list?
(Solution: queue and stack)

37).List different types of data structures?
(Solution: Link list, queue, stack, trees, files, graphs)

.