Maharaja Education Trust® Mysore

MAHARAJA INSTITUTE OF TECHNOLOGY THANDAVAPURA

NH 766, Nanjangud Taluk, Mysuru - 571 302

(An ISO 9001:2015 and ISO 21001:2018 Certified Institution)

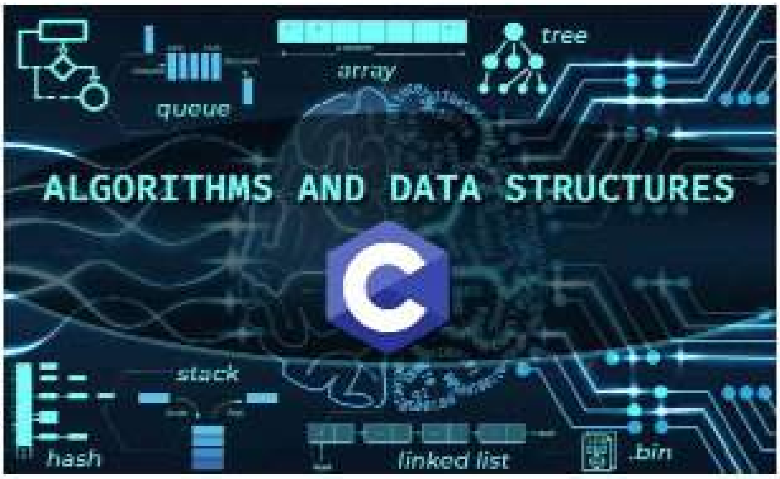
(Affiliated to VTU, Belagavi and approved by AICTE, New Delhi)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

"DATA STRUCTURES AND APPLICATIONS LABORATORY" (18CSL38)

(2018 CBCS Scheme)

As per VTU Revised Syllabus for III Semester CSE



Name: ______________________________________

USN: ____

Batch: _________ Sem: _________ Section: ____________

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

VISION OF THE DEPARTMENT

To impart quality education for producing world class engineers with latest knowledge and innovative ideas in Computer Science & Engineering to meet the expectations of industry and society and to produce globally competent graduates with moral values committed to build a vibrant nation.

MISSION OF THE DEPARTMENT

M 1: To promote technical proficiency by adopting effective teaching learning processes.

M 2: To pursue trending and emerging technologies in Computer Science and Engineering and learn their application across disciplines in order to serve the needs of industry, government, society, and the scientific community.

M 3: To provide environment & opportunity for students to bring out their inherent talents for their all round development.

M 4: To educate students to be successful, ethical, and effective problem solvers and lifelong learners who will contribute positively to the society.

M 5: To make computer science and engineering department a learning and agile centre to nurture the spirit of innovation, creativity and entrepreneurship among the students and faculty.

PROGRAMME SPECIFIC OUTCOMES (PSOs)

On completion of B.E Computer Science & Engineering Program, The graduates will be able to -

PSO 1 The ability to understand, analyse and develop computer programs in the areas related to algorithms, system software, multimedia, web design, big data analytics, and networking for efficient design of computer-based systems of varying complexity.

PSO 2 The ability to understand the evolutionary changes in computing, apply standard practices and strategies in software project development using open-ended programming environments to deliver a quality product for business success, real world problems and meet the challenges of the future.

PSO 3 The ability to employ modern computer languages, environments, and platforms in creating innovative career paths to be an entrepreneur, lifelong learning and a zest for higher studies and also to act as a good citizen by inculcating in them moral values & ethics.

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

In support of the mission, within few years of graduation, the Computer Science & Engineering programme will enable its graduates to -

PEO 1 To be able to comprehend, understand and analyze Computer Science and Engineering problems and relate them with real life.

PEO 2 To provide in depth knowledge to design and develop novel products and innovative solution for real life problems in Computer Science and Engineering field and related domains.

PEO 3 To inculcate a conviction to believe in self, impart professional and ethical attitude, nurture to be an effective team member, infuse leadership qualities, build proficiency in soft skills and the abilities to relate engineering with the social issues.

PEO 4 To impart exhaustive knowledge of Computer Science & Engineering to take up key assignments in industry, undertake and excel in higher studies and Research & Development in computer science, related engineering fields and management.

PROGRAM OUTCOMES (POs)

PO 1: **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO 2: **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO 3: **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO 4: **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide validconclusions.

PO 5: **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO 6: **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineeringpractice.

PO 7: **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainabledevelopment.

PO 8: **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO 9: **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO 10: **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO 11: **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one"s own work, as a memberand leader in a team, to manage projects and in multidisciplinaryenvironments.

PO 12: **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

SYLLABUS

Subject Code	: 18 CSL 38	**CIE Marks**	: 40
Number of Lecture Hours/Week	: 0:2:2	**SEE Marks**	: 60
Total Number of Lab Contact Hours	: 36	**Exam Hours**	: 03

CREDITS - 02

Course Learning Objectives: This course (18CSL38) will enable students to:

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

- Asymptotic performance of algorithms.
- Linear data structures and their applications such as stacks, queues and lists
- Non-Linear data structures and their applications such as trees and graphs
- Sorting and searching algorithms

Descriptions (if any):

Implement all the programs in 'C / C++' Programming Language and Linux / Windows as OS

Program List:

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

	a. Insert an Element on to Circular QUEUE b. Delete an Element from Circular QUEUE c. Demonstrate Overflow and Underflow situations on Circular QUEUE d. Display the status of Circular QUEUE e. Exit Support the program with appropriate functions for each of the above operations
7	Design, Develop and Implement a menu driven Program in C for the following operations on Singly Linked List (SLL) of Student Data with the fields: *USN, Name, Branch, Sem, PhNo* a. Create a SLL of N Students Data by using *front insertion*. b. Display the status of SLL and count the number of nodes in it c. Perform Insertion / Deletion at End of SLL d. Perform Insertion / Deletion at Front of SLL(Demonstration of stack) e. Exit
8	Design, Develop and Implement a menu driven Program in C for the following operations on Doubly Linked List (DLL) of Employee Data with the fields: *SSN, Name, Dept, Designation, Sal, PhNo* a. Create a DLL of N Employees Data by using *end insertion*. b. Display the status of DLL and count the number of nodes in it c. Perform Insertion and Deletion at End of DLL d. Perform Insertion and Deletion at Front of DLL e. Demonstrate how this DLL can be used as Double Ended Queue. f. Exit
9	Design, Develop and Implement a Program in C for the following operations on Singly Circular Linked List (SCLL) with header nodes a. Represent and Evaluate a Polynomial $P(x,y,z) = 6x^2y^2z-4yz^5+3x^3yz+2xy^5z-2xyz^3$ 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	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. Exit
11	Design, Develop and Implement a Program in C for the following operations on Graph(G) of Cities a. Create a Graph of N cities using Adjacency Matrix. b. Print all the nodes reachable from a given starting node in a digraph using DFS/BFS method.
12	Given a File of N employee records with a set K of Keys (4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are Integers. Design and develop a Program in C that uses Hash function H: K ®L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.
Laboratory Outcomes: The student should be able to:	
• Analyze and Compare various linear and non-linear data structures	

- Code, debug and demonstrate the working nature of different types of data structures and their applications
- Implement, analyze and evaluate the searching and sorting algorithms
- Choose the appropriate data structure for solving real world problems

Conduct of Practical Examination:

- All laboratory experiments, excluding the first, are to be included for practical examination.
- Experiment distribution
  - For questions having only one part: Students are allowed to pick one experiment from the lot and are given equal opportunity.
  - For questions having part A and B: Students are allowed to pick one experiment from part A and one experiment from part B and are given equal opportunity.
- Change of experiment is allowed only once and marks allotted for procedure part to be made zero.
- Marks Distribution *(Subjected to change in accordance with university regulations)*

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

  b) For questions having part A and B

     i. Part A – Procedure + Execution + Viva = 4 + 21 + 5 = 30 Marks

     ii. Part B – Procedure + Execution + Viva = 10 + 49+ 11 = 70 Marks

CONTENTS

Introduction

Data Structures:

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

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

Data Structures can be classified as:

Primitive data structures

Non-Primitive data structures.

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

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

Arrays, stacks, queues, trees, graphs etc.

Data structures are also classified as

- Linear data structures
- Non-Linear data structures.

In the Linear data structures processing of data items is possible in linear fashion, i.e., data can be processed one by one sequentially.

Example of such data structures are:

- Array
- Linked list
- Stacks
- Queues

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

Data structure operations:

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

I. Traversing.
II. Searching.
III. Inserting.
IV. Deleting.
V. Sorting.
VI. Merging

Stacks:

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

Queues:

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

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

Linked list:

Disadvantages of static/sequential allocation technique:

1) If an item has to be deleted then all the following items will have to be moved by one allocation. Wastage of time.

2) Inefficient memory utilization.

3) If no consecutive memory (free) is available, execution is not possible.

Linear Linked Lists

Types of Linked lists:

1) Single Linked lists
2) Circular Single Linked Lists
3) Double Linked Lists
4) Circular Double Linked Lists.

Node:

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

The entire linked list is accessed from an external pointer FIRST that points to (contains the address of) the first node in the list. (By an "external" pointer, we mean, one that is not included within a node. Rather its value can be accessed directly by referencing a variable). The nodes in the list can be accessed using a pointer variable. In the above fig. FIRST is the pointer having the address of the first node of the list, initially before creating the list, as list is empty. The FIRST will always be initialized to NULL in the beginning. Once the list is created, FIRST contains the address of the first node of the list.

As each node is having only one link/next, the list is called single linked list and all the nodes are linked in one direction. Each node can be accessed by the pointer pointing (holding the address) to that node, Say P is pointer to a particular node, then the information field of that node can be accessed using info(P) and the next field can be accessed using next(P).

The arrows coming out of the next field in the fig. indicates that the address of the succeeding node is stored in that field.

The link field of last node contains a special value known as NULL which is shown using a diagonal line pictorially. This NULL pointer is used to signal the end of a list.

The basic operations of linked lists are Insertion, Deletion and Display. A list is a dynamic data structure. The number of nodes on a list may vary dramatically as elements are inserted and deleted(removed). The dynamic nature of list may be contrasted with the static nature of an array, whose size remains constant. When an item has to inserted, we will have to create a node, which has to be got from the available free memory of the computer system, So we shall use a mechanism to find an unused node which makes it available to us. For this purpose we shall use the getnode operation (getnode() function). The C language provides the built-in functions like malloc(), calloc(), realloc() and free(), which are stored in alloc.h or stdlib.h header files. To dynamically allocate and release the memory locations from/to the computer system.

Trees:

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

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

Graph:

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

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

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

Graph representation can be done using adjacency matrix.

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

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

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

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

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

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

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

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

Where,

k is the key

D is Max size of hash table.

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

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

PROGRAM – 1

Design, Develop and Implement a menu driven Program in C for the following Array operations

a. Creating an Array of N Integer Elements
b. Display of Array Elements with Suitable Headings
c. Inserting an Element (ELEM) at a given valid Position (POS)
d. Deleting an Element at a given valid Position (POS)
e. Exit.

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

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

int List[SIZE],n=0;

void Create()
{
    int i;
    printf("\n\n\tEnter the Number of Elements : ");
    scanf("%d",&n);
    if(n>SIZE)
        printf("\n\n\tNumber of Elements Exceeds MAX size of
array. . .\n\n");
    else
    {
        printf("\n\n\tEnter %d Elements :-",n);
        for(i=0; i<n; i++)
            scanf("%d",&List[i]);
    }
    Display();

}

void Display()
{
    int i;
    if(n==0)
        printf("\n\n\tList is Empty. . .");
    else
    {
        printf("\n\n\tList Contains : ");
        for(i=0; i<n; i++)
            printf(" %d",List[i]);
    }
}

void Insert()
{
    int Elem,Pos,i;
    if(n == SIZE)
```

```
        printf("\n\n\tList is Full. . .");
    else
    {
        printf("\n\n\tEnter the Element to be Inserted : ");
        scanf("%d",&Elem);
        printf("\n\n\tEnter Position to be Inserted at : ");
        scanf("%d",&Pos);
        if(Pos>0 && Pos<=n+1)
        {
            for(i=n; i>=Pos; i--)
                List[i] = List[i-1];
            List[Pos-1] = Elem;
            n++;
        }
        else
        {
            printf("\n\n\tThe     Given     Position     is
Invalid. . .");
        }
    }
    Display();
}

void Delete()
{
    int Pos,Item,i;
    if(n == 0)
        printf("\n\n\tList is Empty. . .");
    else
    {
        printf("\n\n\tEnter  Position  of  element  to  be
Deleted : ");
        scanf("%d",&Pos);
        if(Pos>0 && Pos<=n)
        {
            Item = List[Pos-1];
            for(i=Pos-1; i<n; i++)
                List[i] = List[i+1];
            n--;
            printf("\n\n\t%d is Removed from the List",Item);
        }
        else
        {
            printf("\n\n\tThe     Given     Position     is
Invalid. . .");
        }
    }
    Display();
}

main()
{
    int Op;
    while(1)
    {
        printf("\n\n\t---------List Menu---------------\n");

printf("\n\n\t1.Create\t2.Insert\t3.Delete\t4.Exit");
        printf("\n\n\tEnter Your Choice : ");
        scanf("%d",&Op);
        switch(Op)
        {
        case 1:
            Create();
            break;
        case 2:
            Insert();
            break;
        case 3:
            Delete();
            break;
        case 4:
            exit(0);
        default:
            printf("\n\n\tInvalid Option. . .\n");
        }
    }
}
```

PROGRAM – 2

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

- **Read a main String (STR), a Pattern String (PAT) and a Replace String (REP)**
- **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.

```
#include<stdio.h>
#define SIZE 1000
int P[SIZE],n=0;

void Read_String(char S[SIZE])
{
    int i=0;
    do
    {
        scanf("%c",&S[i]);
        i++;
    }
    while(S[i-1]!='\n');
    S[i-1] = '\0';
}

int String_Length(char S[SIZE])
{
    int i=0;
    while(S[i]!='\0')
        i++;
    return i;
}

void Find(char STR[SIZE],char PAT[SIZE])
{
    int SLen,PLen,i,j,k;
    SLen = String_Length(STR);
    PLen = String_Length(PAT);
    i=0;
    while(i<=SLen-PLen)
    {
        j = 0;
        k = i;
        while(PAT[j] != '\0')
        {
            if(STR[k] != PAT[j])
                break;
            j++;
            k++;
        }
        if(j == PLen)
        {
            P[n] = i;
            n++;
            i = i + PLen;
        }
        else
            i++;
    }
}

void Replace(char STR[SIZE],char PAT[SIZE],char REP[SIZE])
{
    int PLen,RLen,SLen,i,j,D,k;
    SLen = String_Length(STR);
    PLen = String_Length(PAT);
    RLen = String_Length(REP);
    if(n == 0)
        printf("\n\n\tPattern  does't  exist  in  the  given
String STR");
    else
    {
        for(i=0; i<n; i++)
        {
            D = RLen - PLen;
            k = P[i];
            while(D > 0)
            {
                for(j=SLen; j>=k; j--)
                    STR[j+1] = STR[j];
                SLen++;
                k++;
                D--;
            }
            k = P[i] + PLen-1;
            while(D < 0)
            {
                for(j=k; j<=SLen; j++)
                    STR[j-1] = STR[j];
                SLen--;
                k--;
                D++;
            }
            j = 0;
            k = P[i];
            while(REP[j] != '\0')
            {
                STR[k] = REP[j];
                k++;
                j++;
            }
            for(j=i+1; j<n; j++)
                P[j] = P[j] + (RLen - PLen);
        }
    }
}

int main()
{
    char STR[100],PAT[10],REP[10];
    printf("\t\t\tString Operation Demonstration\n");
    printf("\t\t\t------------------------------\n");
    printf("\n\n Enter a String : ");
    Read_String(STR);
    printf("\n\n Enter a Pattern : ");
    Read_String(PAT);
    printf("\n\n Enter a Replace String : ");
    Read_String(REP);
    Find(STR,PAT);
    Replace(STR,PAT,REP);
    printf("\n\n Updated Main String STR is %s\n\n\n",STR);
    return 0;
}
```

PROGRAM - 3

Design, Develop and Implement a menu driven Program in C for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX)

a. Push an Element 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

```c
#include<stdio.h>
#include<stdlib.h>
#define MAX 5
int Stack[MAX], Top = -1;

void Display()
{
    int i;
    if(Top == -1)
        printf("\n\n\tStack is Empty");
    else
    {
        printf("\n\n\tStack Contains:-\n");
        for(i=Top; i>=0; i--)
            printf("\t\t%d\n",Stack[i]);
    }
}

void Push(int Item)
{
    if(Top == MAX-1)
        printf("\n\n\tStack Overflow");
    else
        Stack[++Top] = Item;
    Display();
}

int Pop()
{
    if(Top == -1)
    {
        printf("\n\n\tStack Underflow");
        return 999;
    }
    else
        return Stack[Top--];
}

void Palindrome()
```

```
{
    int Num,Digit[MAX],D,i,j;
    printf("\n\n\tEnter a Number : ");
    scanf("%d",&Num);
    Top = -1;
    j=0;
    while(Num>0)
    {
        D = Num % 10;
        Digit[j++] = D;
        Push(D);
        Num = Num / 10;
    }
    for(i=0; i<j; i++)
    {
        if(Digit[i] != Pop())
            break;
    }
    if(Top == -1)
        printf("\n\n\tThe Given Number is Palindrome");
    else
        printf("\n\n\tThe Given Number is not a Palindrome");
}

int main()
{
    int Option,Num;
    while(1)
    {
            printf("\n---------Stack Menu---------------");
        printf("\n1-->Push\t\t2-->Pop\n3-->Palindrome\t4-
->Exit");
        printf("\n---------------------------------------");
        printf("\n\n\tEnter Your Choice : ");
        scanf("%d",&Option);
        switch(Option)
        {
        case 1:
            printf("\n\n\tEnter a Number : ");
            scanf("%d",&Num);
            Push(Num);
            break;
        case 2:
            Num = Pop();
            if(Num != 999)
            {
                printf("\n\n\t%d   is   Popped   from   the
Stack",Num);
                Display();
            }

            break;
        case 3:
                Palindrome();
                break;
            case 4:
                exit(0);
                default: printf("\n\n\tInvalid Choice. . .");
            }
        }
    return 0;
}
```

PROGRAM - 4

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

```
#include<stdio.h>
#define MAX 100
char Stack[MAX];
int Top = -1;

void Push(char Ch)
{
    Stack[++Top] = Ch;
}

char Pop()
{
    return Stack[Top--];
}

int Priority(char Ch)
{
    int P;
    switch(Ch)
    {
    case '#' :
        P = 0;
        break;
    case '(' :
        P = 1;
        break;
    case '+' :
    case '-' :
        P = 2;
        break;
    case '*' :
    case '/' :
    case '%' :
        P = 3;
        break;
    case '^' :
        P = 4;
        break;
    }
    return P;
}

void Infix_To_Postfix(char Infix[MAX], char Postfix[MAX])
{
    int i,j;
    char Ch;
    Push('#');
    i = 0;
    j = 0;
    while(Infix[i]!='\0')
    {
        switch(Infix[i])
        {
        case '(' :
            Push(Infix[i]);
            break;
        case ')' :
            Ch = Pop();
            while(Ch != '(')
            {
                Postfix[j++] = Ch;
                Ch = Pop();
            }
            break;
        case '+' :
        case '-' :
        case '*' :
        case '/' :
        case '%' :
        case '^' :
           while(Priority(Stack[Top]) >= Priority(Infix[i]))
                Postfix[j++] = Pop();
            Push(Infix[i]);
            break;
        default:
            Postfix[j++] = Infix[i];
        }
        i++;
    }
    while(Top > 0)
        Postfix[j++] = Pop();
    Postfix[j] = '\0';
}

int main()
{
    char Infix[MAX], Postfix[MAX];
    printf("\n\n\tEnter the Valid Infix Expression : ");
    scanf("%s",Infix);
    Infix_To_Postfix(Infix,Postfix);
    printf("\n\n\tPostfix Expression is %s\n\n\n",Postfix);
    return 0;
}
```

PROGRAM - 5A

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

```
#include<stdio.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 '%':
          return (int)op1%(int)op2;
     case '^':
          return pow(op1,op2);
     default:
          return 0;
     }
}
void main()
{
     double s[20],res,op1,op2;
     int top,i;
     char postfix[20],symbol;
     printf("\n\n\tEnter the postfix expression:");
     gets(postfix);
     top=-1;
     for(i=0; i<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("\n\n\tThe result is:%.2f\n",res);
}
```

PROGRAM - 5B

Design, Develop and Implement a Program in C for solving the Tower of Hanoi problem with n disks

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

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

void main()
{
    int n;
    printf("\n\n\t------------TOWER OF HANOI------------");
    printf("\n\n\t Enter the number of disks:");
    scanf("%d",&n);
    tower(n,'A','B','C');
    printf("\n\n\tTotal        number        of        moves
are:%d",(int)pow(2,n)-1);
}
```

PROGRAM - 6

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

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

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

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 5

char Q[MAX];
int Rear = -1, Front = 0, Count = 0;

void Insert(char Item)
{
    if(Count == MAX)
        printf("\n\n\tQueue Overflow. . .");
    else
    {
        Rear = ( Rear+1 ) % MAX;
        Q[Rear] = Item;
        Count++;
    }
}
char Delete()
{
    char Item;
    if(Count == 0)
    {
        printf("\n\n\tQueue Underflow. . .");
        return '\0';
    }
    else
    {
        Item = Q[Front];
        Front = ( Front+1 ) % MAX;
        Count--;
        return Item;
    }
}
void Display()
{
    int i,j;
    if(Count == 0)
        printf("\n\n\tQueue is Empty. . .");
    else
    {
        printf("\n\n\tQueue Contains : ");
        j = Front;
        for(i=0; i<Count; i++)
        {
            printf("%c ",Q[j]);
            j = (j+1) % MAX;
        }
    }
}

int main()
{
    int Option;
    char Ch[0];
    while(1)
    {
        printf("\n-------------Queue Menu--------------");
        printf("\n  1  -->  Insert\t\t  2  -->  Delete\n  3  -->
Exit");
        printf("\n-------------------------------------");
        printf("\n\n\tEnter Your Choice : ");
        scanf("%d",&Option);
        switch(Option)
        {
        case 1:
            printf("\n\n\tEnter a Character : ");
            scanf("%s",Ch);
            Insert(Ch[0]);
            Display();
            break;
        case 2:
            Ch[0] = Delete();
            if(Ch[0] != '\0')
                printf("\n\n\t%c   is   Removed   from   the
Queue",Ch[0]);
            Display();
            break;
        case 3:
            exit(0);
        }
    }
    return 0;
}
```

PROGRAM - 7

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

a. Create a SLL of N Students Data by using front insertion.
b. Display the status of SLL and count the number of nodes in it
c. Perform Insertion / Deletion at End of SLL
d. Perform Insertion / Deletion at Front of SLL(Demonstration of stack)
e. Exit

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

typedef struct NODE
{
    char USN[11];
    char Name[30];
    char Branch[5];
    int Sem;
    char PhNo[11];
    struct NODE *next;
} node;

node *Start, *New;

void Create_Node()
{
    New = (node*)malloc(sizeof(node));
    printf("\n\n\tEnter the Details of the Student :-\n");
    printf("\n\n\tUSN : ");
    scanf("%s",New->USN);
    printf("\n\n\tName : ");
    scanf("%s",New->Name);
    printf("\n\n\tBranch : ");
    scanf("%s",New->Branch);
    printf("\n\n\tSemester : ");
    scanf("%d",&New->Sem);
    printf("\n\n\tPhone Number : ");
    scanf("%s",New->PhNo);
    New->next = NULL;
}

void Insert_at_Front()
{
    Create_Node();
    if(Start==NULL)
        Start = New;
    else
    {
        New->next = Start;
        Start = New;
    }
}

void Insert_at_End()
{
    Create_Node();
    node *Temp;
    if(Start==NULL)
        Start = New;
    else
    {
        Temp = Start;
        while(Temp->next != NULL)
            Temp = Temp->next;
        Temp->next = New;
    }
}

void Delete_at_Front()
{
    node *Temp;
    if(Start==NULL)
        printf("\n\tList is Empty, cannot Delete. . .");
    else
    {
        Temp = Start;
        Start = Start->next;
        printf("\n\n\tNode    with    USN    :    %s    is
Deleted",Temp->USN);
        free(Temp);
    }
}

void Delete_at_End()
{
    node *Temp, *Prev;
    if(Start==NULL)
        printf("\n\n\tList is Empty, cannot Delete. . .");
    else
    {
        if(Start->next == NULL)
        {
            printf("\n\n\tNode   with   USN   :   %s   is
Deleted",Start->USN);
            free(Start);
            Start = NULL;
        }
        else
        {
            Temp = Start;
            Prev = Temp;
            while(Temp->next != NULL)
            {
                Prev = Temp;
                Temp = Temp->next;
            }
            Prev->next = NULL;
            printf("\n\n\tNode    with    USN    :    %s    is
Deleted",Temp->USN);
            free(Temp);
        }
    }
}

void Display()
{
    int Count = 0;
    node *Temp;
    if(Start==NULL)
        printf("\n\n\n\tList is Empty");
    else
    {
        printf("\n\n\n\tSTART-->");
        Temp = Start;
        while(Temp!=NULL)
        {
            printf("\n\n\n\t[%s|%s|%s|%d|%s]-
->",Temp->USN,Temp->Name,

     Temp->Branch,Temp->Sem,Temp->PhNo);
            Count++;
            Temp = Temp->next;
        }
        printf("NULL");
    }
    printf("\n\n\tList Contains %d Nodes",Count);
}

void At_End()
{
    int Option;
    while(1)
    {
        printf("\n\n\n\n-----Operations at End Menu------");
        printf("\n\n1->Insertion\t2->Deletion\t3->Return  to
Main Menu");
        printf("\n\n------------------------------------");
        printf("\n\n\tEnter Your Option : ");
        scanf("%d",&Option);
        switch(Option)
        {
        case 1:
            Insert_at_End();
            break;
        case 2:
            Delete_at_End();
```

```
                break;
            case 3:
                return;
            }
    }
}

void At_Front()
{
    int Option;
    while(1)
    {
        printf("\n\n\n\n-----Operations at Front Menu----");
        printf("\n\n1->Insertion\t2->Deletion\t3->Return  to
Main Menu");
        printf("\n\n------------------------------------");
        printf("\n\n\tEnter Your Option : ");
        scanf("%d",&Option);
        switch(Option)
        {
        case 1:
            Insert_at_Front();
            break;
        case 2:
            Delete_at_Front();
            break;
        case 3:
            return;
        }
    }
}

void Stack()
{
    int Option;
    while(1)
    {
        printf("\n\n\n\n-----Stack Operations Menu------");
        printf("\n\n1->Push\t2->Pop\t3->Return            to
Demonstration Menu");
        printf("\n\n------------------------------------");
        printf("\n\n\tEnter Your Option : ");
        scanf("%d",&Option);
        switch(Option)
        {
        case 1:
            Insert_at_End();
            Display();
            break;
        case 2:
            if(Start == NULL)
                printf("\n\n\tStack is Empty, Cannot Pop");
            else
```

```
                {
                    Delete_at_End();
                    Display();
                }
                break;
            case 3:
                return;
            }
    }
}

void Demo()
{
    int Option;
    while(1)
    {
        printf("\n\n\n\n--------Demonstration Menu-------");
        printf("\n\n 1 --> Stack\t2 --> Return to Main Menu");
        printf("\n\n---------------------------------------");
        printf("\n\n\tEnter Your Option : ");
        scanf("%d",&Option);
        switch(Option)
        {
        case 1:
            Stack();
            break;
        case 2:
            return;
        }
    }
}

void main()
{
    int Option,N,i;
    while(1)
    {
        printf("\n\n\n\n-----Singly Linked List Menu-----");
        printf("\n\n1->Create N Students List\t2->Operations
at End");
            printf("\t3->Operations                         at
Front\n\n4->Demo\t\t\t5->Exit");
        printf("\n\n---------------------------------------");
        printf("\n\n\tEnter Your Option : ");
        scanf("%d",&Option);
        switch(Option)
        {
        case 1:
            Start = NULL;
            printf("\n\n\tEnter Number of Students : ");
            scanf("%d",&N);
            for(i=0; i<N; i++)
                Insert_at_Front();
```

```
                    Display();
                break;
            case 2:
                At_End();
                Display();
                break;
            case 3:
                At_Front();
                Display();
                break;
            case 4:
                Demo();
                break;
            case 5:
                exit(0);
            }
        }
}
```

PROGRAM - 8

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

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

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

typedef struct NODE
{
    int SSN;
    char Name[30];
    char Dept[15];
    char Design[15];
    float Salary;
    char PhNo[11];
    struct NODE *next;
    struct NODE *prev;
} node;
node *Start, *New;

void Create_Node()
{
    New = (node*)malloc(sizeof(node));
    printf("\n\n\tEnter the Details of the Employee :-\n");
    printf("\n\n\tSSN: ");
    scanf("%d",&New->SSN);
    printf("\n\n\tName: ");
    scanf("%s",New->Name);
    printf("\n\n\tDepartment: ");
    scanf("%s",New->Dept);
    printf("\n\n\tDesignation : ");
    scanf("%s",New->Design);
    printf("\n\n\tSalary: ");
    scanf("%f",&New->Salary);
    printf("\n\n\tPhone Number: ");
    scanf("%s",New->PhNo);
    New->next = NULL;
    New->prev = NULL;
}

void Insert_at_Front()
{
    Create_Node();
    if(Start==NULL)
        Start = New;
    else
    {
        New->next = Start;
        Start->prev = New;
        Start = New;
    }
}

void Insert_at_End()
{
    Create_Node();
    node *Temp;
    if(Start==NULL)
        Start = New;
    else
    {
        Temp = Start;
        while(Temp->next != NULL)
            Temp = Temp->next;
        Temp->next = New;
        New->prev = Temp;
    }
}

void Delete_at_Front()
{
    node *Temp;
    if(Start==NULL)
        printf("\n\tList is Empty, cannot Delete");
    else
    {
        Temp = Start;
        Start = Start->next;
        Start->prev = NULL;
        printf("\n\n\tNode    with    SSN    :    %d    is
Deleted",Temp->SSN);
        free(Temp);
    }
}

void Delete_at_End()
{
    node *Temp, *Prev;
    if(Start==NULL)
        printf("\n\n\tList is Empty, cannot Delete");
    else
    {
        if(Start->next == NULL)
        {
            printf("\n\n\tNode    with    SSN    :    %d    is
Deleted",Start->SSN);
            free(Start);
            Start = NULL;
        }
        else
        {
            Temp = Start;
            Prev = Temp;
            while(Temp->next != NULL)
            {
                Prev = Temp;
                Temp = Temp->next;
            }
            Prev->next = NULL;
            printf("\n\n\tNode    with    SSN    :    %d    is
Deleted",Temp->SSN);
            free(Temp);
        }
    }
}

void Display()
{
    int Count = 0;
    node *Temp;
    if(Start==NULL)
        printf("\n\n\n\tList is Empty");
    else
    {
        printf("\n\n\n\tNULL<-->");
        Temp = Start;
        while(Temp!=NULL)
        {
            printf("[%d|%s|%s|%s|%f|%s]-
->",Temp->SSN,Temp->Name,

     Temp->Dept,Temp->Design,Temp->Salary,Temp->PhNo);
            Count++;
            Temp = Temp->next;
        }
        printf("NULL");
    }
    printf("\n\n\tList Contains %d Nodes",Count);
}

void At_End()
{
    int Option;
    while(1)
    {
        printf("\n\n\n\n----Operations at End Menu-------");
        printf("\n\n1->Insertion\t2->Deletion\t3->Return  to
Main Menu");
```

```
        printf("\n\n------------------------------------");
        printf("\n\n\tEnter Your Option : ");
        scanf("%d",&Option);
        switch(Option)
        {
        case 1:
            Insert_at_End();
            break;
        case 2:
            Delete_at_End();
            break;
        case 3:
            return;
        }
    }
}

void At_Front()
{
    int Option;
    while(1)
    {
        printf("\n\n\n\n-----Operations at Front Menu----");
        printf("\n\n1->Insertion\t2->Deletion\t3->Return  to 
Main Menu");
        printf("\n\n------------------------------------");
        printf("\n\n\tEnter Your Option : ");
        scanf("%d",&Option);
        switch(Option)
        {
        case 1:
            Insert_at_Front();
            break;
        case 2:
            Delete_at_Front();
            break;
        case 3:
            return;
        }
    }
}

void DQueue()
{
    int Option;
    while(1)
    {
        printf("\n\n\n\n-----DQueue Operations Menu------");
        printf("\n\n1->Insert    Front\t2->Insert    Rear\t
3->Delete Front");
        printf("\n\t\t4->Delete  Rear\t5->  Return  to  Main 
Menu");
        printf("\n\n------------------------------------");
        printf("\n\n\tEnter Your Option : ");
        scanf("%d",&Option);
        switch(Option)
        {
        case 1:
            Insert_at_Front();
            Display();
            break;
        case 2:
            Insert_at_End();
            Display();
            break;
        case 3:
            if(Start == NULL)
                printf("\n\n\tDQueue   is   Empty,   Cannot
Delete");
            else
            {
                Delete_at_Front();
                Display();
            }
            break;
        case 4:
            if(Start == NULL)
                printf("\n\n\tDQueue   is   Empty,   Cannot
Delete");
            else
            {
                Delete_at_End();
                Display();
            }
            break;
        case 5:
            return;
        }
    }
}
void main()
{
    int Option,N,i;
    while(1)
    {
        printf("\n\n\n\n-----Doubly Linked List Menu-----");
        printf("\n\n1->Create          N          Employees
List\t\t2->Display");
        printf("\n\n3->Operations at End\t\t4->Operations at
Front");
        printf("\n\n5->Demo DQueue\t\t\t6->Exit");
        printf("\n\n------------------------------------");
        printf("\n\n\tEnter Your Option : ");
        scanf("%d",&Option);
        switch(Option)
        {
```

```
        case 1:
            Start = NULL;
            printf("\n\n\tEnter Number of Employees : ");
            scanf("%d",&N);
            for(i=0; i<N; i++)
                Insert_at_End();
            break;
        case 2:
            Display();
            break;
        case 3:
            At_End();
            break;
        case 4:
            At_Front();
            break;
        case 5:
            DQueue();
            break;
        case 6:
            exit(0);
        }
    }
}
```

PROGRAM - 9

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

a. Represent and Evaluate a Polynomial $P(x,y,z)=6x^2y^2z-4yz^5+3x^3yz+2xy^5z-2xyz^3$

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

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

typedef struct NODE
{
    int Coef;
    int ExpoX;
    int ExpoY;
    int ExpoZ;
    struct NODE *next;
    int Check;
} node;
node *Start=NULL, *New,*POLY1,*POLY2,*POLYSUM;

void Create_Poly(int C,int Ex, int Ey, int Ez)
{
    node *Temp;
    New = (node*)malloc(sizeof(node));
    New->Coef  = C;
    New->ExpoX = Ex;
    New->ExpoY = Ey;
    New->ExpoZ = Ez;
    New->next  = New;
    New->Check = 0;
    if(Start == NULL)
        Start = New;
    else
    {
        Temp = Start;
        while(Temp->next != Start)
            Temp = Temp->next;
        Temp->next = New;
        New->next = Start;
    }
}

void Read_Poly()
{
    int C,Ex,Ey,Ez,n,i;
    printf("\n\n\tEnter Number of Terms : ");
    scanf("%d",&n);
    for(i=0; i<n; i++)
    {
        printf("\n\n\tEnter the Coefficient : ");
        scanf("%d",&C);
        printf("\n\n\tEnter the Exponent of Variable X : ");
        scanf("%d",&Ex);
        printf("\n\n\tEnter the Exponent of Variable Y : ");
        scanf("%d",&Ey);
        printf("\n\n\tEnter the Exponent of Variable Z : ");
        scanf("%d",&Ez);
        Create_Poly(C,Ex,Ey,Ez);
    }
}

void Display(node *Start)
{
    node *Temp = Start;
    printf("\n\n Start-->");
    do
    {

printf("[%dx^%dy^%dz^%d]->",Temp->Coef,Temp->ExpoX,Temp->Exp
oY,
                                Temp->ExpoZ);
        Temp = Temp->next;
    }
    while(Temp!=Start);
    printf(" Start");
}
void Eval_Poly()
{
    int Result=0,x,z,y;
    node *Temp;
    printf("\n\n\tEnter the Value of Variable X : ");
    scanf("%d",&x);
    printf("\n\n\tEnter the Value of Variable Y : ");
    scanf("%d",&y);
    printf("\n\n\tEnter the Value of Variable Z : ");
    scanf("%d",&z);
    Temp = POLY1;
    do
    {

Result=Result+Temp->Coef*pow(x,Temp->ExpoX)*pow(y,Temp->Expo
Y)* pow(z,Temp->ExpoZ);
        Temp = Temp->next;
    }
    while(Temp!=POLY1);
    printf("\n\n\tResult : %d",Result);
}

void Add_Poly()
{
    node *Temp1,*Temp2;
    Start = NULL;
    Temp1 = POLY1;
    do
    {
        Temp2 = POLY2;
        do
        {
if((Temp1->ExpoX==Temp2->ExpoX)&&(Temp1->ExpoY==Temp2->ExpoY
)
&&(Temp1->ExpoZ==Temp2->ExpoZ)&&(Temp2->Check == 0))
            {

Create_Poly(Temp1->Coef+Temp2->Coef,Temp1->ExpoX,

Temp1->ExpoY,Temp1->ExpoZ);
                Temp1->Check = 1;
                Temp2->Check = 1;
                break;
            }
            Temp2 = Temp2->next;
        }
        while(Temp2 != POLY2);
        if(Temp2 == POLY2 && Temp1->Check == 0)
        {

Create_Poly(Temp1->Coef,Temp1->ExpoX,Temp1->ExpoY,
                                        Temp1->ExpoZ);
            Temp1->Check = 1;
        }
        Temp1 = Temp1->next;
    }
    while(Temp1 != POLY1);
    Temp2 = POLY2;
    do
    {
        if(Temp2->Check == 0)
        {

Create_Poly(Temp2->Coef,Temp2->ExpoX,Temp2->ExpoY,
                                        Temp2->ExpoZ);
            Temp2->Check = 1;
        }
        Temp2 = Temp2->next;
    }
    while(Temp2 != POLY2);
    POLYSUM = Start;
}
void main()
{
    int Option;
    while(1)
    {
        printf("\n\n-------Polynomial Menu---------");
        printf("\n1->Evaluate  Polynomial\t2->Addition  of  2
Polynomial");
            printf("\t3->Exit");
        printf("\n\n--------------------------------");
        printf("\n\n\tEnter Your Option : ");
        scanf("%d",&Option);
        switch(Option)
        {
        case 1:
            printf("\n\nEnter a Polynomial :-");
            Start = NULL;
            Read_Poly();
            POLY1 = Start;
            Eval_Poly();
            break;
        case 2:
            printf("\n\nEnter 1st Polynomial :-");
            Start = NULL;
            Read_Poly();
            POLY1 = Start;
            printf("\n\nEnter 2nd Polynomial :-");
            Start = NULL;
            Read_Poly();
            POLY2 = Start;
            Add_Poly();
            printf("\n\n\n The 1st Polynomial is :");
            Display(POLY1);
            printf("\n\n\n The 2nd Polynomial is :");
            Display(POLY2);
            printf("\n\n\n The Resultant Polynomial is :");
            Display(POLYSUM);
            break;
        case 3:
            exit(0);
        }
    }
}
```

PROGRAM - 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. Exit

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

typedef struct Node
{
    int Data;
    struct Node *Left;
    struct Node *Right;
} node;
node *New;

void Create_Node()
{
    New = (node*)malloc(sizeof(node));
    printf("\n\n\tEnter the Data of the Node : ");
    scanf("%d",&New->Data);
    New->Left = NULL;
    New->Right = NULL;
}

node* Insert(node* Temp)
{
    if (Temp == NULL)
        return New;
    if (New->Data < Temp->Data)
        Temp->Left = Insert(Temp->Left);
    else if (New->Data > Temp->Data)
        Temp->Right = Insert(Temp->Right);
    return Temp;
}

void PreOrder(node *Temp)
{
    if (Temp != NULL)
    {
        printf("%d ",Temp->Data);
        PreOrder(Temp->Left);
        PreOrder(Temp->Right);
    }
}
```

```
void InOrder(node *Temp)
{
    if (Temp != NULL)
    {
        InOrder(Temp->Left);
        printf("%d ",Temp->Data);
        InOrder(Temp->Right);
    }
}

void PostOrder(node *Temp)
{
    if (Temp != NULL)
    {
        PostOrder(Temp->Left);
        PostOrder(Temp->Right);
        printf("%d ",Temp->Data);
    }
}

int Search(node *Temp,int Key)
{
    if(Temp == NULL)
        return 0;
    if(Temp->Data == Key)
        return 1;
    if(Key < Temp->Data)
        Search(Temp->Left,Key);
    if(Key > Temp->Data)
        Search(Temp->Right,Key);
}

void main()
{
    int Option,n,i,Key;
    node *Root = NULL;
    while(1)
    {
        printf("\n\n--------------BST Menu-------------");
        printf("\n1->Create
BST\t2->PreOrder\t3->InOrder\t4->PostOrder");
        printf("\n\n5->Search\t6->Exit");
        printf("\n\n-----------------------------------");
        printf("\n\n\tEnter Your Option:");
        scanf("%d",&Option);
        switch(Option)
        {
        case 1:
            printf("\n\n\tEnter Number of Elements : ");
            scanf("%d",&n);
            for(i=0; i<n; i++)
            {
                Create_Node();
                Root = Insert(Root);
            }
            break;
        case 2:
            printf("\n\n\tPreOrder Traversing : ");
            PreOrder(Root);
            break;
        case 3:
            printf("\n\n\tInOrder Traversing : ");
            InOrder(Root);
            break;
        case 4:
            printf("\n\n\tPostOrder Traversing : ");
            PostOrder(Root);
            break;
        case 5:
            printf("\n\n\tEnter the Key Element : ");
            scanf("%d",&Key);
            if(Search(Root,Key))
                printf("\n\n\t%d is Found",Key);
            else
                printf("\n\n\t%d is Not Found",Key);
            break;
        case 6:
            exit(0);
        }
    }
}
```

PROGRAM - 11

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

a. Create a Graph of N cities using Adjacency Matrix.

b. Print all the nodes reachable from a given starting node in a digraph using DFS/BFS method

```
#include<stdio.h>
#include<stdlib.h>
int G[10][10],src,visit[10],n;

void Graph_read()
{
    int i,j;
    printf("Enter the number of vertices in the graph:");
    scanf("%d",&n);
    printf("Enter the Graph as Adjacency Matrix:\n");
    for(i=1; i<=n; i++)
        for(j=1; j<=n; j++)
            scanf("%d",&G[i][j]);
}

void bfs(int src, int visit[10])
{
    int f=0,r=0;
    int q[10],u,v,i;
    q[r]=src;
    visit[src]=1;
    while(f<=r)
    {
        u=q[f++];
        for(v=1; v<=n; v++)
        {
            if(G[u][v]==1 && visit[v]==0)
            {
                visit[v]=1;
                q[++r]=v;
            }
        }
    }
}

void dfs(int src)
{
    int i;
    visit[src]=1;
    for(i=1; i<=n; i++)
    {
        if(visit[i]==0 && G[src][i]==1)
            dfs(i);
    }
}
```

```
void isconnected(int flag)
{
    if(flag==1)
        printf("Graph is not connected\n");
    else
        printf("Graph is connected\n");
}

void main()
{
    int i,j,src,ch,flag;
    while(1)
    {
        printf("\n----------------Graph Menu---------------");
        printf("\n1.Create  Graph  of  Matrix\t2.Depth  First
Search\t");
            printf("\t3.Breadth First Search\t4.Exit");
        printf("\n---------------------------------------------");
        printf("\n\nEnter your Choice:");
        scanf("%d",&ch);
        switch(ch)
        {
        case 1:
            Graph_read();
            break;
        case 2:
            printf("Enter the source vertex:");
            scanf("%d",&src);
            for(i=1; i<=n; i++)
                visit[i]=0;
            bfs(src,visit);
            flag=0;
            for(i=1; i<=n; i++)
            {
                if(visit[i]==0)
                {
                    printf("Node       %d       is       not
reachable...\n",i);
                    flag=1;
                }
                else
                    printf("Node %d is reachable...\n",i);
            }
            isconnected(flag);
            break;
        case 3:
            for(i=1; i<=n; i++)
                visit[i]=0;
            printf("Enter the source vertex:");
            scanf("%d",&src);
            dfs(src);
            flag=0;
            for(i=1; i<=n; i++)
            {
                if(visit[i]==0)
                {
                    printf("Node        %d        is        not 
reachable...\n",i);
                    flag=1;
                }
                else
                    printf("Node %d is reachable...\n",i);
            }
            isconnected(flag);
            break;
        case 4:
            exit(0);
        }
    }

}
```

PROGRAM - 12

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

```
#include<stdio.h>
#define MAX 5

int HashTable[MAX];

struct Employee
{
    int Key;
    char Name[25];
    char Desig[20];
    float Salary;
} E[MAX];

void Create_Hash_Table(int N)
{
    int i,j,Count;
    for(i=0; i<MAX; i++)
        HashTable[i] = -1;
    for(i=0; i<N; i++)
    {
        j = E[i].Key % MAX;
        Count = 0;
        while(HashTable[j]!=-1)
        {
            j = (j+1)%MAX;
            Count++;
        }
        HashTable[j] = i;
    }
    printf("\n\n\tHash Table:-\n\n");
    for(i=0; i<MAX; i++)
    {
        if(HashTable[i]!=-1)
        {
            printf("\nHashTable[%d]->Record of Employee with
Key-%d\n",i,
                                    E[HashTable[i]].Key);
            printf("%d\t%s\t%s\t%f\n",E[HashTable[i]].Key,

     E[HashTable[i]].Name,E[HashTable[i]].Desig,
                                      E[HashTable[i]].Salary);
        }
        else
            printf("\nHashTable[%d] -> Empty\n",i);
    }
}


int Read_File()
{
    FILE *fp;
    int i;
    fp = fopen("Emp.txt","r");
    i = 0;
    while(!feof(fp))
    {
        if(i==MAX)
        {
            printf("Hash Table is Full...Cannot Read anymore
Record!!!");
            break;
        }

fscanf(fp,"%d%s%s%f\n",&E[i].Key,E[i].Name,E[i].Desig,
                        &E[i].Salary);
        i++;
    }
    return i;
}

void main()
{
    int N;
    N = Read_File();
    Create_Hash_Table(N);
}
```

VIVA QUESTIONS

Q.No.	Questions and Answers
1.	**What is data structure?**
A:	A data structure is a way of organizing data that considers not only the items stored, but also their relationship to each other. Advance knowledge about the relationship between data items allows designing of efficient algorithms for the manipulation of data.
2.	**List out the areas in which data structures are applied extensively?**
A:	1. Compiler Design 2. Operating System 3. Database Management System 4. Statistical analysis package 5. Numerical Analysis 6. Graphics 7. Artificial Intelligence 8. Simulation
3.	**What are the major data structures used in the following areas : RDBMS, Network data model and Hierarchical data model.**
A:	1. RDBMS = Array (i.e. Array of structures) 2. Network data model = Graph 3. Hierarchical data model = Trees
4.	**If you are using C language to implement the heterogeneous linked list, what pointer type will you use?**
A:	The heterogeneous linked list contains different data types in its nodes and we need a link, pointer to connect them. It is not possible to use ordinary pointers for this. So we go for void pointer. Void pointer is capable of storing pointer to any type as it is a generic pointer type.
5.	**Minimum number of queues needed to implement the priority queue?**
A:	Two. One queue is used for actual storing of data and another for storing priorities.
6.	**What is the data structures used to perform recursion?**
A:	Stack. Because of its LIFO (Last In First Out) property it remembers its 'caller' so knows whom to return when the function has to return. Recursion makes use of system stack for storing the return addresses of the function calls. Every recursive function has its equivalent iterative (non-recursive) function. Even when such equivalent iterative procedures are written, explicit stack is to be used.
7.	**What are the notations used in Evaluation of Arithmetic Expressions using prefix and postfix forms?**
A:	Polish and Reverse Polish notations.
8.	**Convert the expression ((A + B) * C - (D - E) ^ (F + G)) to equivalent Prefix and Postfix notations.**
A:	1. Prefix Notation: ^ - * +ABC - DE + FG 2. Postfix Notation: AB + C * DE - - FG + ^

Q.No.	Questions and Answers
9.	**Sorting is not possible by using which of the following methods? (Insertion, Selection, Exchange, Deletion)**
A:	Sorting is not possible in Deletion. Using insertion we can perform insertion sort, using selection we can perform selection sort, using exchange we can perform the bubble sort (and other similar sorting methods). But no sorting method can be done just using deletion.
10.	**What are the methods available in storing sequential files?**
A:	1. Straight merging, 2. Natural merging, 3. Polyphase sort, 4. Distribution of Initial runs.
11.	**List out few of the Application of tree data-structure?**
A:	1. The manipulation of Arithmetic expression 2. Symbol Table construction 3. Syntax analysis.
12.	**List out few of the applications that make use of Multilinked Structures?**
A:	1. Sparse matrix, 2. Index generation.
13.	**In tree construction which is the suitable efficient data structure? (Array, Linked list, Stack, Queue)**
A:	Linked list is the suitable efficient data structure.
14.	**What is the type of the algorithm used in solving the 8 Queens problem?**
A:	Backtracking.
15.	**In an AVL tree, at what condition the balancing is to be done?**
A:	If the 'pivotal value' (or the 'Height factor') is greater than 1 or less than -1.
16.	**What is the bucket size, when the overlapping and collision occur at same time?**
A:	One. If there is only one entry possible in the bucket, when the collision occurs, there is no way to accommodate the colliding value. This results in the overlapping of values.
17.	**Classify the Hashing Functions based on the various methods by which the key value is found.**
A:	1. Direct method, 2. Subtraction method, 3. Modulo-Division method, 4. Digit-Extraction method, 5. Mid-Square method, 6. Folding method, 7. Pseudo-random method.

Q.No.	Questions and Answers
18.	**What are the types of Collision Resolution Techniques and the methods used in each of the type?**
A:	1. Open addressing (closed hashing), The methods used include: Overflow block. 2. Closed addressing (open hashing), The methods used include: Linked list, Binary tree.
19.	**In RDBMS, what is the efficient data structure used in the internal storage representation?**
A:	B+ tree. Because in B+ tree, all the data is stored only in leaf nodes, that makes searching easier. This corresponds to the records that shall be stored in leaf nodes.
20.	**What is a spanning Tree?**
A:	A spanning tree is a tree associated with a network. All the nodes of the graph appear on the tree once. A minimum spanning tree is a spanning tree organized so that the total edge weight between nodes is minimized.
21.	**Does the minimum spanning tree of a graph give the shortest distance between any 2 specified nodes?**
A:	The Minimal spanning tree assures that the total weight of the tree is kept at its minimum. But it doesn't mean that the distance between any two nodes involved in the minimum spanning tree is minimum.
22.	**Which is the simplest file structure? (Sequential, Indexed, Random)**
A:	Sequential is the simplest file structure.
23.	**Whether Linked List is linear or Non-linear data structure?**
A:	• According to Access strategies Linked list is a linear one. • According to Storage Linked List is a Non-linear one.
24.	**Define a stack?**
A:	Stack is an ordered collection of elements. Insertion and deletion of stack is done only from one end, called top of the stack. It is also called LIFO(Last in first out)
25.	**Explain a Queue?**
A:	Queues are first in first out type(FIFO). New elements are added to the one end called REAR and the elements are removed from the other end is called FRONT. The people standing in railway reservation queue is a practical example of queue
26.	**What are the different types of queues available?**
A:	Linear Queue, Circular Queue, Priority Queue
27.	**What is the main difference between a stack and a queue?**
A:	Stack is LIFO and queue is FIFO. In stack insertion and deletion happening from one end(top of the stack). But in Queue neew elements are added to the one end called REAR and the elements are removed from the other end is called FRONT

Q.No.	Questions and Answers
28.	**What are the applications of Queue?**
A:	Round robin technique for processor scheduling , all types of customer services like railway ticket reservation center, printer server routines can be implemented using queue.
29.	**What are linear and nonlinear data Structures?**
A:	Linear: A data structure is said to be linear if its elements form a sequence or a linear list. Examples: Array. Linked List, Stacks and Queues Non-Linear: A data structure is said to be non-linear if traversal of nodes is nonlinear in nature. Example: Graph and Trees.
30.	**What are the various operations that can be performed on different Data Structures?**
A:	Insertion: Add a new data item in the given collection of data items. Deletion: Delete an existing data item from the given collection of data items. Traversal: Access each data item exactly once so that it can be processed. Searching: Find out the location of the data item if it exists in the given collection of data items. Sorting: Arranging the data items in some order i.e. in ascending or descending order in case of numerical data and in dictionary order in case of alphanumeric data.
31.	**How is an Array different from Linked List?**
A:	• The size of the arrays is fixed, Linked Lists are Dynamic in size. • Inserting and deleting a new element in an array of elements is expensive, Whereas both insertion and deletion can easily be done in Linked Lists. • Random access is not allowed in Linked Listed. • Extra memory space for a pointer is required with each element of the Linked list. • Arrays have better cache locality that can make a pretty big difference in performance
32.	**What is Stack and where it can be used?**
A:	Stack is a linear data structure which the order LIFO(Last In First Out) or FILO(First In Last Out) for accessing elements. Basic operations of stack are : Push, Pop , Display Applications of Stack: Infix to Postfix Conversion using Stack Evaluation of Postfix Expression Reverse a String using Stack Implement two stacks in an array Check for balanced parentheses in an expression
33.	**What is a Queue, how it is different from stack and how is it implemented?**
A:	Queue is a linear structure which follows the order is First In First Out (FIFO) to access elements. Mainly the following are basic operations on queue: Enqueue, Dequeue, Front, Rear

Q.No.	Questions and Answers
	The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added. Both Queues and Stacks can be implemented using Arrays and Linked Lists.
34.	**What are Infix, prefix, Postfix notations?**
A:	Infix notation: X + Y – Operators are written in-between their operands. This is the usual way we write expressions. An expression such as A * (B + C) / D Postfix notation (also known as "Reverse Polish notation"): X Y + Operators are written after their operands. The infix expression given above is equivalent to A B C + * D/ Prefix notation (also known as "Polish notation"): + X Y Operators are written before their operands. The expressions given above are equivalent to / * A + B C D
36.	**Which data structures are used for BFS and DFS of a graph?**
A:	Queue is used for BFS Stack is used for DFS. DFS can also be implemented using recursion (Note that recursion also uses function call stack).
37.	**What is a Linked List and What are its types?**
A:	A linked list is a linear data structure (like arrays) where each element is a separate object. Each element (that is node) of a list is comprising of two items – the data and a reference to the next node.Types of Linked List: Singly Linked List : In this type of linked list, every node stores address or reference of next node in list and the last node has next address or reference as NULL. For example 1->2->3->4->NULL Doubly Linked List : Here, here are two references associated with each node, One of the reference points to the next node and one to the previous node. Eg. NULL<-1<->2<->3->NULL Circular Linked List : Circular linked list is a linked list where all nodes are connected to form a circle. There is no NULL at the end. A circular linked list can be a singly circular linked list or doubly circular linked list. Eg. 1->2->3->1 [The next pointer of last node is pointing to the first]
38.	**How to implement a stack using queue?**
A:	A stack can be implemented using two queues. Let stack to be implemented be 's' and queues used to implement be 'q1' and 'q2'. Stack 's' can be implemented in two ways: Method 1 (By making push operation costly) Method 2 (By making pop operation costly) See Implement Stack using Queues
39.	**We use two data structures to implement an LRU Cache.**
A:	• Queue which is implemented using a doubly linked list. The maximum size of the queue will be equal to the total number of frames available (cache size).The most recently used pages will be near rear end and least recently pages will be near front end.

Q.No.	Questions and Answers
	• A Hash with page number as key and address of the corresponding queue node as value.
40.	**Differentiate between file and structure storage structure.**
A:	The key difference between both the data structure 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.
41.	**When is a binary search best applied?**
A:	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 searched 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.
42.	**What is a linked list?**
A:	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 for data storage.
43.	**How do you reference all the elements in a one-dimension array?**
A:	To reference all the elements in a one -dimension array, you need to use an indexed loop, So that, the counter runs from 0 to the array size minus one. In this manner, You can reference all the elements in sequence by using the loop counter as the array subscript.
44.	**In what areas do data structures are applied?**
A:	Data structures are essential 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.
43.	**What is LIFO?**
A:	LIFO is a short form of Last In First Out. It refers 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.
44.	**What is a queue?**
A:	A queue is a data structure that can simulate 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.
45.	**What are binary trees?**
A:	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 an extension of the linked list structures.

Q.No.	Questions and Answers
46.	**Which data structures are applied when dealing with a recursive function?**
A:	Recursion, is 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.
47.	**What is a stack?**
A:	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.
48.	**Explain Binary Search Tree**
A:	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.
49.	**What are multidimensional arrays?**
A:	Multidimensional arrays make use of multiple indexes to store data. It is useful when storing data that cannot be represented using single dimensional indexing, such as data representation in a board game, tables with data stored in more than one column.
50.	**Are linked lists considered linear or non-linear data structures?**
A:	It 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.
51.	**How does dynamic memory allocation help in managing data?**
A:	Apart 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.
52.	**What is FIFO?**
A:	FIFO stands 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.
53.	**What is an ordered list?**
A:	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 values form an increasing sequence, as the list is traversed.
54.	**What is merge sort?**
A:	Merge sort, is a divide-and-conquer approach for sorting the 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 continues until you have one single sorted list.

Q.No.	Questions and Answers
55.	**Differentiate NULL and VOID**
A:	Null is a value, whereas Void is a data type identifier. A variable that is given a Null value indicates an empty value. The void is used to identify pointers as having no initial size.
56.	**What is the primary advantage of a linked list?**
A:	A linked list is an ideal data structure because it can be modified easily. This means that editing a linked list works regardless of how many elements are in the list.
57.	**What is the difference between a PUSH and a POP?**
A:	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.
58.	**What is a linear search?**
A:	A linear search refers to the way a target key is being searched in a sequential data structure. In this method, each element in the list is checked and compared against the target key. The process is repeated until found or if the end of the file has been reached.
59.	**How does variable declaration affect memory allocation?**
A:	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.
60.	**What is the advantage of the heap over a stack?**
A:	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, the memory of the heap can at times be slower when compared to that stack.
61.	**What is a postfix expression?**
A:	A postfix expression is an expression in which each operator follows its operands. The advantage of this form is that there is no need to group sub-expressions in parentheses or to consider operator precedence.
62.	**What is Data abstraction?**
A:	Data abstraction is a powerful tool for breaking down complex data problems into manageable chunks. This is applied by initially specifying the data objects involved and the operations to be performed on these data objects without being overly concerned with how the data objects will be represented and stored in memory.