JAIN COLLEGE OF ENGINEERING, BELAGAVI

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
(ACADEMIC YEAR 2024-25)

LABORATORY MANUAL

SUBJECT: DATA STRUCTURES LABORATORY
SUB CODE: BCSL305

SEMESTER: III-2022 CBCS Scheme

Prepared By
Prof. Nalinakshi
B.G

Approved By
Dr.Uttam Patil.
HOD, CSE

Program Outcomes (POs)

Engineering Graduates will be able to:

PO1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2. **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.

PO3. **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.

PO4. **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.

PO5. **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.

PO6. **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.

PO7. **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.

PO8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10. **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.

PO11. **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.

PO12. 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.

Program Specific Outcomes (PSOs)

PSO1: Apply mathematical and scientific skills in the area of computer science and engineering to design, develop, analyze software and hardware based systems.

PSO2: Provide solutions using networking and database administration skills, and address the needs of environmental and societal issues through entrepreneurial practices.

Program Educational Objectives (PEOs):

1. To produce graduates who have a strong foundation of knowledge and Engineering skills that will enable them to have successful career in the field of computer science and engineering.
2. To expose graduates to professional and team building skills along with ideas of innovation and invention.
3. To prepare graduates with an ethics, social responsibilities, and professional concerns.

DATA STRUCTURES LABORATORY
(Effective from the academic year 2022 -2026)
SEMESTER – III

Subject Code	BCSL305	**CIE Marks**	50
Number of Contact Hours/Week	0:0:2	**SEE Marks**	50
Total Number of Lab Contact Hours	28	**Exam Hours**	3 Hrs.

Credits – 2

Course Learning Objectives:

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

- Dynamic memory management.
- Linear data structures and their applications such as stacks, queues and lists
- Non-Linear data structures and their applications such as trees and graphs

Descriptions (if any):

- Implement all the programs in "C " Programming Language and Linux OS

Programs List:

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.
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 c. Support the program with functions for each of the above operations. Don't use Built-in functions.
3.	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.	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 alphanumericoperands.
5.	Develop 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.	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.	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.	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. a. Exit
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
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
11.	Develop 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. 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 various linear and non-linear data structures
- Demonstrate the working nature of different types of data structures and their applications
- Use appropriate searching and sorting algorithms for the give scenario.
- Apply the appropriate data structure for solving real world problems

Conduct of Practical Examination:

- Experiment distribution
    - For laboratories having only one part: Students are allowed to pick one experiment from the lot with equal opportunity.
    - For laboratories having PART A and PART B: Students are allowed to pick one experiment from PART A and one experiment from PART B, with equal opportunity.
- Change of experiment is allowed only once and marks allotted for procedure to be made zero of the changed part only.
- Marks Distribution *(Need to change in accordance with university regulations)*
    - For laboratories having only one part – Procedure + Execution + Viva-Voce: 15+70+15 = 100 Marks
    - For laboratories having PART A and PART B
        i. Part A – Procedure + Execution + Viva = 6 + 28 + 6 = 40 Marks
        ii. Part B – Procedure + Execution + Viva = 9 + 42 + 9 = 60 Marks

CONTENTS

Sl.No.	EXPERIMENT NAME
1.	Introduction
2.	**Program 1 :** Array Operations
3.	**Program 2 :** String Operations
4.	**Program 3 :** Stack Operations
5.	**Program 4** : Infix to Postfix Conversion
6.	**Program 5:** Design, Develop and Implement Program in C for the following Stack Applications a. Evaluation of Suffix expression with single digit operands and operators: +, -, *, /,%, ^. b. Solving Tower of Hanoiproblemwithndisks
7.	**Program 6 :** Circular Queue Operations
8.	**Program 7 :** Implementation of Singly Linked List
9.	**Program 8 :** Implementation of Doubly Linked List
10.	**Program 9 :** Polynomial Evaluation & Addition using SCLL with header node
11.	**Program 10 :** Implementation of Binary Search tree
12.	**Program 11 :** Implementation of Graphs (BFS & DFS Methods)
13.	**Program 12 :** Implementation of Hashing & Linear Probing

Introduction to Data Structure

Basic Concepts

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.

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.1 2 3 4 5

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.

APPLICATION OF QUEUE

Queue, as the name suggests is used whenever we need to have any group of objects in an order in which the first one coming in, also gets out first while the others wait for their turn, like in the following scenarios :

Serving requests on a single shared resource, like a printer, CPU task scheduling etc.

In real life, Call Center phone systems will use Queues, to hold people calling them in an order, until a service representative is free.

Handling of interrupts in real-time systems. The interrupts are handled in the same order as they arrive, First come first served.

LINKED LIST

Disadvantages of static/sequential allocation technique:

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

Inefficient memory utilization.

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

Linear Linked Lists

Types of Linked lists:

- Single Linked lists
- Circular Single Linked Lists
- Double Linked Lists
- 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. Rath accessed directly by referencing a variable).

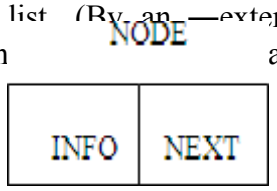


Fig-1 Linked List

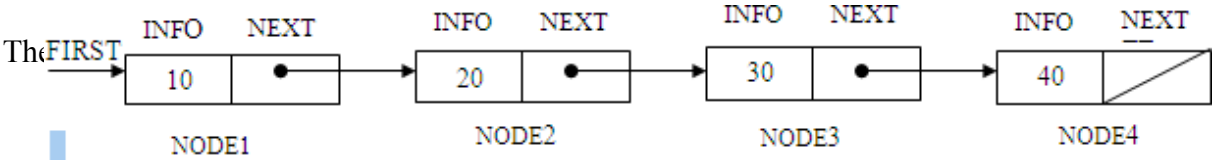


Fig-2 Linked List

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.

Tree is a non-linear data structure which organizes data in hierarchical structure and this is a recursive definition.

A tree data structure can also be defined as follows...

Tree data structure is a collection of data (Node) which is organized in hierarchical structure and this is a recursive definition

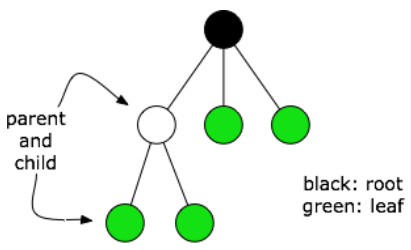


Figure: tree data structure

Binary tree: A tree with utmost two children for each node.

Complete Binary Tree: A binary tree in which every level, except possibly the deepest, is completely filled. At depth n, the height of the tree, all nodes must be as far left as possible. Binary search tree: A binary tree where every node‘s left subtree has keys less than the node's key, and every right subtree has keys greater than the node's key.

Tree traversal is a technique for processing the 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.

PROGRAM 1

Develop a Program in C for the following:

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

Program objective:

- Understand the working of arrays and structures.
- Understand the working of dynamic memory allocation.

Algorithm:

Step 1: Start.

Step 2: Read number of Calendar

days. Step 3: Read the details for each

day.

Step 4: Details read are Day, Date and Activity Description.

Step 5: Print the Weeks activity details.

Step 7: Stop

THEORY:

Array is a collection of elements of the same type. Arrays are the kind of data structure that can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

In C, memory allocation can be categorized into two types: static and dynamic. Static memory allocation occurs at compile-time and is determined by the size of variables declared in the source code. Dynamic memory allocation, on the other hand, takes place during program execution, enabling the allocation and deallocation of memory as needed.

Declaring Arrays
To declare an array in C, a programmer specifies the type of the elements and the number of elements required by an array as follows –
type array Name [arraySize];

Accessing Array Elements
An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array.
For example − double salary = balance [9];
The above statement will take the 10^{th} element from the array and assign the value to salary variable.

Dynamic Memory Functions
malloc (Memory Allocation):
The malloc function is used to allocate a specified number of bytes of memory from the heap.
Syntax: void *malloc (size_t size);
Returns a pointer to the first byte of the allocated memory or NULL if the allocation fails.

Other Functions include, calloc (Contiguous Allocation), realloc (Reallocation)

free (Deallocation):
The free function is used to release the dynamically allocated memory.
Syntax: void free(void *ptr);
Once memory is freed, it can be reused for other purposes.

PROGRAM:

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

// Structure to represent a day
struct Day {
    char *name;         // Day of the
    week int date;  // Date of the day
    char *description; // Activity description
};
// Function to create a day
struct Day create() {
    struct Day day;
    char temp[100];
    printf("Enter day of the week:
    "); scanf("%s", temp);
    day.name = strdup(temp); // Dynamically allocate memory for the name

    printf("Enter date: ");
    scanf("%d", &day.date);

    printf("Enter activity description: ");
    scanf(" %[^\n]s", temp); // Read the entire line including spaces
    day.description = strdup(temp); // Dynamically allocate memory for the description

    return day;
}

// Function to read and populate the calendar
void read(struct Day *calendar, int numDays) {
    int i;
    for (i = 0; i < numDays; i++) {
        printf("\nEnter details for Day %d:\n", i + 1);
```

```
    calendar[i] = create();
  }
```

```
// Function to display the calendar
void display(const struct Day *calendar, int numDays) {
    printf("\nCalendar:\n");
    int i;
    for (i = 0; i < numDays; i++) {
        printf("Day %d: %s\n", i + 1, calendar[i].name);
        printf("Date: %d\n", calendar[i].date);
        printf("Activity: %s\n", calendar[i].description);
        printf("\n");
    }
}

int main() {
    int numDays;
    printf("Enter the number of days in the calendar:
    "); scanf("%d", &numDays);
    // Dynamically allocate memory for the calendar
    struct Day *calendar = (struct Day *)malloc(numDays * sizeof(struct Day));

    if (calendar == NULL) {
        printf("Memory allocation failed. Exiting...\n");
        return 1;
    }

    read(calendar, numDays);
    display(calendar, numDays);

    // Free dynamically allocated
    memory int i;
    for (i = 0; i < numDays; i++) {
        free(calendar[i].name);
        free(calendar[i].description)
        ;
    }
    free(calendar);
```

```
return 0;
```

Output

Enter the number of days in the calendar: 2
Enter details for Day 1:
Enter day of the week: Wednesday
Enter date: 02092024
Enter activity description: Assignment Submission
Enter details for Day 2:
Enter day of the week: Friday
Enter date: 03092024
Enter activity description: Project Implementation
Calendar:
Day 1: Wednesday
Date: 01092024
Activity: Assignment
Submission Day 2: Friday
Date: 03092024
Activity: Project Implementation

Program outcome:

- Implement the C program using arrays, structures and dynamic memory.
- Familiarized with the usage of structure initialization.
- Familiarized with the usage of dynamic memory allocation and de-allocation.

Viva Questions:

- What is an array? How to access elements of array?
- Can you change size of array once created?
- What is the difference between arrays and structures?
- What is dynamic memory? Which part of the memory it is allocated?
- Why the memory allocated dynamically must be freed?

PROGRAM 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

Program objective:

- Understand the implementation of string function‘s using arrays.
- Understand pattern matching algorithm and the implementation technique ofthe same without using built-in functions.
- Understand the pattern replacement methodology.

Algorithm:

Step 1: Start.

Step 2: Read main string STR, pattern string PAT and replace string REP.

Step 3: compare pattern string in main string,

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.

THEORY

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. 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.

PROGRAM:

```
#include<stdio.h>
void main()
{
 char
 s[20],pat[20],rep[20],ans[30]; int
 i,j,k,l,flag,found; printf("\nEnter
 string:");
 gets(s);
 printf("\nEnter
 pattern:"); gets(pat);
 printf("\nEnter
 replacement:"); gets (rep);
 found=0;
 for(i=0,k=0;s[i]!='\0';i++
 )
 {
  flag=1;
  for(j=0;pat[j]!='\0';j++
  )
   if(s[i+j]!=pat[j])
    flag=0;
  l=j;
  if(flag
  )
  {
   for(j=0;rep[j]!='\0';j++,k++
    ) ans[k]=rep[j];
   i+=l-1;
   found=1;
  }
  else
   ans[k++]=s[i];
 }
 ans[k]='\0';
```

```
  if (found==0)
    printf("Pattern not found in the main
  string\n"); else
    printf("Modified string after replacement: %s\n", ans);
}
```

Output 1

Enter string:Good Morning JCER

Enter pattern:Morning

Enter replacement:Afternoon

Modified string after replacement: Good Afternoon JCER

Output 2

Enter string:Good Morning JCER

Enter pattern:Afternoon

Enter replacement:Evening

Pattern not found in the main string

Program outcomes:

- Implement string matching and string replacement algorithm without usingbuilt-in library functions.
- Apply the knowledge of array usage to implement string functions.
- Identify different applications of string matching and string replacement.

Viva Questions:

- What is a string?
- How strings are represented in C language? What does strlen do in C?
- Is there a string data type in C? What is the use of char in C programming?

PROGRAM 3

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

a. Push an Element onto 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.

Program objective:

- Understand the concept of palindrome.
- Understand the stack data structures.
- Understand the different functions on stacks i.e., push, pop and implement thesame.
- Understand stack overflow and underflow.

Algorithm:

PUSH (item)
Step 1: Read an element to be pushed on to stack item
Step 2: check overflow condition of stack before inserting element into stack Top=max-1
Step 3: update the top pointer and insert an element into stack
Top=top+1
S[top] <-item

POP (item)
Step1: check underflow condition of stack before deleting element from stack top=-1
Step2: Display deleted element pointed by top
Deleted element<- s[top]
Step3: Decrement top pointer by 1 top<-top-1

Palindrome

Step 1: Two pointers are required , one is pointed to top of stack
another is bottom of stack

Step 2: compare top and bottom elements of stack if it is equal update top and bottom pointer by1

Step 3: if all elements are equal, then stack content is palindrome

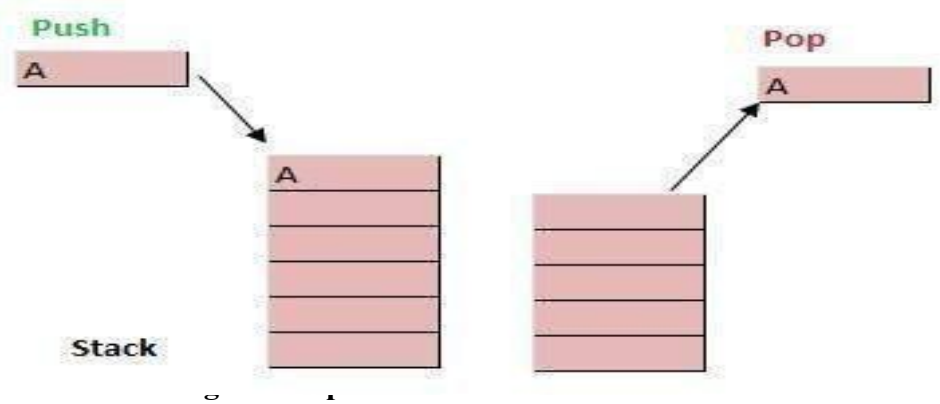
THEORY

It is called as last in, first out. The element inserted first is the last one to be deleted. It is used for various applications like infix to postfix expression, postfix evaluation and for maintaining stack frames for function calling

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 be a 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 performed on stack:

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;

peek() − get the top data element of the stack, without removing it.

isFull() − check if stack is full.

isEmpty() − check if stack is empty.

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#define MAXSIZE 5
int s[10],top=-1;

void push()
{
   if(top==MAXSIZE-1)
      printf("\nStack overflow!!!!");
   else
   {
      printf("\nEnter element to insert:");
      scanf("%d",&s[++top]);
   }
}

void pop()
{
   if(top==-1)
      printf("\nStack underflow!!!");
   else
      printf("\nElement popped is: %d",s[top--]);
}
void disp()
{
   int
   t=top;
   if(t==-1)
      printf("\nStack
   empty!!"); else
      printf("\nStack elements are:\n");
   while(t>=0)
      printf("%d ",s[t--]);
}
void pali()
{
   int num[5],rev[5],i,t;
   for(i=0,t=top;t>=0;i++,t--
   )
      num[i]=rev[t]=s[t];
```

```
  for(i=0;i<=top;i++)
      if(num[i]!=rev[i]
      ) break;
```

```
    if(i==top+1)
        printf("\nIt is a
    palindrome"); else
        printf("\nIt is not a palindrome");
}

int main()
{
    int
    ch; do
    {
        printf("\n...Stack operations     \n");
        printf("1.PUSH\n");
        printf("2.POP\n");
        printf("3.Palindrome\n")
        ; printf("4.Display\n");
        printf("5.Exit\n_____
        \n"); printf("Enter choice:");
        scanf("%d",&ch);
        switch(ch)
        {
            case
            1:push();break;
            case 2:pop();break;
            case 3:pali();break;
            case 4:disp();break;
            case 5:exit(0);
            default:printf("\nInvalid choice");
        }
    }
    while(1)
    ; return
    0;

}
```

Output

```
...Stack operations.....
1.PUSH
2.POP_____
3.      Palind
rome
4.Display
```

5.Exit

Enter choice:1

Enter element to insert:10

...Stack operations.....

1.PUSH
2.POP
3. Palind
rome
4.Display
5.Exit

Enter choice:1

Enter element to insert:20

...Stack operations.....
1.PUSH
2.POP
3. Palind
rome
4.Display
5.Exit

Enter choice:1

Enter element to insert:30

...Stack operations.....
1.PUSH
2.POP
3. Palind
rome
4.Display
5.Exit

Enter choice:1

Enter element to insert:40

...Stack operations.....
1.PUSH
2.POP
3. Palind
rome
4.Display
5.Exit

Enter choice:1

Enter element to insert:50

...Stack operations.....

1.PUSH
2.POP
3. Palind
rome
4.Display
5.Exit

Enter choice:1

Stack overflow!!!!
...Stack operations.....
1.PUSH
2.POP
3. Palind
rome
4.Display
5.Exit

Enter choice:2

Element popped is: 50
...Stack operations.....
1.PUSH
2.POP
3. Palind
rome
4.Display
5.Exit

Enter choice:2

Element popped is: 40
...Stack operations.....
1.PUSH
2.POP
3. Palind
rome
4.Display
5.Exit

Enter choice:2

Element popped is: 30

...Stack operations.....

1.PUSH

2.POP

3. Palindrome

4.Display

5.Exit

Enter choice:3

It is not a palindrome
...Stack operations.....
1.PUSH
2.POP
3. Palind rome
4.Display
5.Exit

Enter choice:4

Stack elements are: 20 10
...Stack operations.....
1.PUSH
2.POP
3. Palind rome
4.Display
5.Exit

Enter choice:5

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.

Program objective:

- Understand different notations to represent regular expression.
- Understand infix to postfix conversion.
- Understand the precedence of operators.

Algorithm:

Step 1: Read the infix expression as a string.

Step 2: Scan the expression character by character till the end. Repeat the following operations

1. If it is an operand add it to the postfixexpression.
2. If it is a left parenthesis push it onto the stack.
3. If it is a right parentheses pop out elements from the stack and assign it to the postfix string. Pop out the left parentheses but don‘t assign topostfix.

Step 3: If it is an operator compare its precedence with that of the element at the top of stack.

1. If it is greater push it onto the stack.
2. Else pop and assign elements in the stack to the postfix expression untilyou find one such element.

Step 4: If you have reached the end of the expression, pop out any leftover elements in the stack till it becomes empty.

Step 5: Append a null terminator at the end display the result

THEORY

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	ABC*+

Infix to prefix conversion Expression = **(A+B^C)*D+E^5**

Step 1. Reverse the infix expression.

5^E+D*)C^B+A(

Step 2. Make Every '(' as ')' and every ')' as '('

5^E+D*(C^B+A)

Step 3. Convert expression to postfix form.

Step 4. Reverse the expression.

+*+A^BCD^E

Step 5. Result

+*+A^BCD^E5

PROGRAM:

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

int F(char symbol)
{
 switch (symbol)
 {
 case '+':
 case '-':return 2;
 case '*':
 case '/':
 case '%':return
 4; case '^':
 case '$':return 5;
 case '(':return 0;
 case '#':return -1;
 default :return 8;
 }
}

int G(char symbol)
{
 switch (symbol)
 {
 case '+':
 case '-':return 1;
 case '*':
 case '/':
 case '%':return
 3; case '^':
 case '$':return 6;
 case '(':return 3;
 case ')':return 0;
```

```
default :return 7;
}
```

```
}

void infix_postfix(char infix[], char postfix[])
{
int top=-1, j=0, i;
char s[30], symbol;
s[++top] = '#';
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];
printf("\nEnter a valid infix expression\n") ;
scanf ("%s", infix) ;
infix_postfix (infix, postfix);
printf("\nThe infix expression
is:\n"); printf ("%s",infix);
printf("\nThe postfix expression is:\n");
printf ("%s",postfix) ;
```

```
}
```

Output:

Enter a valid infix expression
((a+b)*c)
The infix expression is:
((a+b)*c)
The postfix expression is:
ab+c*

Program outcome :

- Identify the applications of infix and postfix.
- Implement C program to convert infix to postfix.
- Identify the different operators.

Viva Questions:

- What is a postfix expression?
- What are Infix, prefix, Postfix notations?
- What is the evaluation order according to which an infixexpressionis converted to postfix expression ?
- which data structure is used for infix to postfix conversion

PROGRAM 5

Design, develop and implement a Program in C for the following Stack Applications

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

b. Solving Tower of Hanoi problem with n disks

Program objective :

- Understand different polish notation.
- Understand the methodology of evaluating suffix expression.
- Get the knowledge of operator preeedenee and assoeiativity.

Algorithm

Step 1: Read the suffix/postfix expression

Step 2: Scan the postfix expression from left to right character by character

Step 3: if scanned symbol is operand push data into stack.

If scanned symbol is operator pop two elements from stack Evaluate result and result is pushed onto stack

Step 4: Repeat step 2-3 until all symbols are scanned completely

PROGRAM:

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

float compute(char symbol, float op1, float 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()
{
float s[20], res, op1, op2;
int top, i;
char postfix[20], symbol;
printf("\nEnter the postfix expression:\n");
scanf ("%s", 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("\nThe result is : %f\n", res);

}
```

Output1

Enter the postfix
expression: 23+
The result is : 5.000000

Output2

Enter the postfix
expression: 123-4*+
The result is : -3.000000

Program outcome:

- Identify the applications of suffix expression.
- Familiarized with the methodology of suffix evaluation.
- Familiarized the operator precedence and associativity.

Viva Questions

- What is Suffix Expression?

5 b. Solving Tower of Hanoi problem with n disks

Program objective:

- Understand tower of Hanoi problem.
- Understand recursive functions and its disadvantages.

Algorithm:

MAIN FUNCTION ()

Step 1: Read No of disks called n from keyboard.
Step 2: Check if n is not zero or a negative no. if yes display suitable message else go to step3.
Step 3: Call tower of Hanoi function with n asparameter,
Step 4: Stop

TOWERS OF HANOI FUNCTION TO MOVE DISKS FROM A TO C USING B ()

Step 1: If n is equal to 1 then move the single disk from A to C and stop Step 2: Move the top n

-

Step 1 disks from A to B using c as auxiliary.
Step 3: Move the remaining disk from A to C.
Step 4: Move the n-1 disks from B to C using as auxiliary.

THEORY

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 program 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 maybe placed on top of a smallerdisk.

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

PROGRAM:

```
#include<stdio.h>
#include<math.h>
void tower(int n, int source, int temp, int destination);

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;
printf("\nEnter the number of discs: \n\n");
scanf("%d", &n);
printf("\nThe sequence of moves involved in the Tower of Hanoi are\n");
tower(n, 'A', 'B', 'C');
printf("\n\nTotal Number of moves are: %d\n", (int)pow(2,n)-1);

}
```

Output

```
Enter the number of discs:
3
The sequence of moves involved in the Tower of Hanoi are
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
```

Program outcome:

- Identify the application of Tower of Hanoi problem.
- Implement the methodology to solve Tower of Hanoi problem.
- Implement the given problem using recursive function.

Viva Questions

- How do you solve the problem of the Tower of Hanoi using recursion?
- What is recursion? And what is tower of Hanoi problem?

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

Program objective:

- Understand the working of circular queue
- Know the advantages of circular queue over liner queue.
- Understand the insertion and deletion operation on circular queue.
- Understand overflow and underflow conditions in circular queue.

ALGORITHM:

Step1: Initialize front and rear pointer and also
count front->0,count<-0,rear<- -1

Step2: Insert an element into queue before check overflow condition
Count=max
Insert an element rear<-(rear+1)
%max q[rear]<-item and
count=count+1

Step3: Delete an element from queue .check underflow condition
Count=0 underflow condition. Count<-count-1
Item<-q[front]Deleted element

Step4: Display contents of queue. Number of elements represents count.
Check empty queue condition before displaying an element

THEORY

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**.

It is also called FIFO structure. 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.

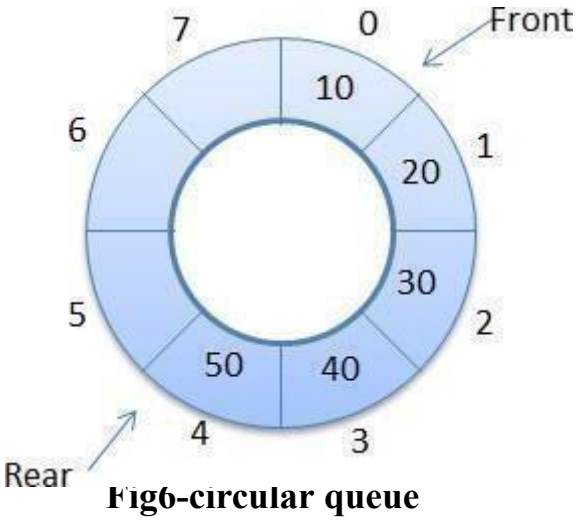


Fig6-circular queue

The **limitation** of **simple queue** is that even if there is a free memory space available in the simple queue we cannot use that free memory space to insert element. **Circular Queue is designed to overcome the limitation of Simple Queue.**

PROGRAM:

```
#include <stdio.h>
#include
<stdlib.h> #define
max 5
int q[max],f=-1,r=-1;
void ins()
{
   if(f==(r+1)%max)
      printf("\nQueue overflow");
   else
   {
      if(f==-1)
         f++;
      r=(r+1)%max;
      printf("\nEnter element to be
      inserted:"); scanf("%d",&q[r]);
   }
}
void del()
{
   if(r==-1)
      printf("\nQueue
   underflow"); else
   {
      printf("\nElemnt deleted
      is:%d",q[f]); if(f==r)
         f=r=-1;
      else
         f=(f+1)%max;
   }
}
void disp()
{
   if(f==-1)
      printf("\nQueue
   empty"); else
   {
      int i;
```

```
printf("\nQueue elements
are:\n");
for(i=f;i!=r;i=(i+1)%max)
   printf("%d\t",q[i]);
printf("%d",q[i]);
printf("\nFront is at:%d\nRear is at:%d",q[f],q[r]);
```

```
    }
}
int main()
{
    printf("\nCircular Queue operations");
    printf("\n1.Insert");
    printf("\n2.Delete");
    printf("\n3.Display");
    printf("\n4.Exit");
    int
    ch;
    do{
        printf("\nEnter
        choice:");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:ins();break;
            case 2:del();break;
            case 3:disp();break;
            case 4:exit(0);
            default:printf("\nInvalid choice...!");
        }
   }while(1);
    return 0;

}
```

Output

```
Circular Queue
operations 1.Insert
2.Delete
3.Displa
y 4.Exit
Enter choice:1
Enter element to be inserted:10
Enter choice:1
Enter element to be inserted:20
Enter choice:1
Enter element to be
inserted:30 Enter choice:1
Enter element to be inserted:40
```

Enter choice:1
Enter element to be inserted:50
Enter choice:1
Queue overflow

Enter choice:3
Queue elements are:
10 20 30 40 50
Front is at:10
Rear is at:50
Enter choice:2
Elemnt deleted is:10 Enter choice:2 Elemnt deleted is:20 Enter choice:2 Elemnt deleted is:30 Enter choice:
2
Elemnt deleted is:40 Enter choice:
2
Elemnt deleted is:50 Enter choice:
2
Queue underflow Enter choice:
4

Program outcome:

- Identify the applications of circular queue.
- Implement insert and delete operations on circular queue.

Viva Questions:

- What is a queue ?what are applications of queue?
- What is Circular Queue? What is the difference between a Stack and a Queue?

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**

Program objective:

- Understand the Singly Linked List (SLL) data structures.
- Understand the methodology to insert and delete the element at the front of SLL.
- Understand the methodology to insert and delete the element at the end ofSLL.
- Get the knowledge of how SLL and be used as both stack and queue.

Algorithm

Step 1: declare structure of node create empty
list head->null
Step2: **Insert at front end**

head<-null
returntem
p
if list is
empty
temp->link=head
return head
Step 3**:Insert at rear end**
head=null
return temp
if list is empty

cur->head
while(cur!=null
) cur=cur->link
cur->link=temp;

return head

Step 4**: Delete at front end**
head->link=null
; return null
if list has only one
node cur=head
head=head->lin
k free(cur)
Step 5:**Delete at Rear end**
head-
>link=n
u ll
retur
n null
if only one
node cur<-head
while(cur!=null
)
prev<-cur, cur=cur<-link;
free(cur);

THEORY

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.

In any single linked list, the individual element is called as "Node". Every "Node" contains two fields, data and next. The data field is used to store actual value of that node and next field is used to store the address of the next node in the sequence.

The graphical representation of a node in a single linked list is as follows...

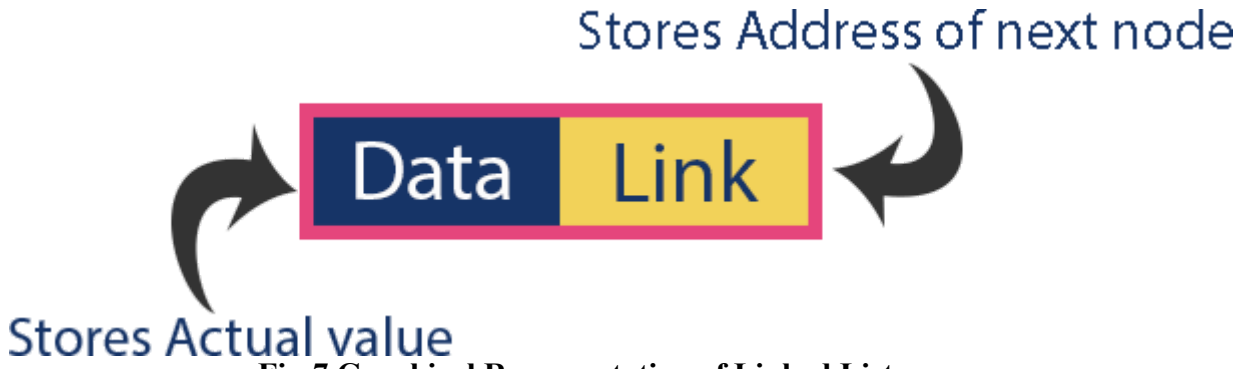


Fig-7 Graphical Representation of Linked List

In a single linked list, the address of the first node is always stored in a reference node known as "front" (Some times it is also known as "head").Always next part (reference part) of the last node must be NULL.

They are a dynamic in nature which allocates the memory when required.

- Insertion and deletion operations can be easilyimplemented.
- 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.

Advantages over arrays

1) Dynamic size
2) Ease of insertion/deletion

Drawbacks:

1) Random access is not allowed. We have to access elements sequentially starting fromthe first node. So we cannot do binary search with linked lists.
2) Extra memory space for a pointer is required with each element of the list.

PROGRAM:

```
#include<string.h
>
#include<stdio.h>
#include<stdlib.h>
struct stud
{
   char usn[11],name[15],branch[4],phno[11];
   int sem;
   struct stud *next;
}*f=NULL,*r=NULL,*t=NULL;
void ins(int ch)
{
   t=(struct stud*)malloc(sizeof(struct
   stud)); printf("\nEnter USN:");
   scanf("%s",t->usn);
   printf("Enter Name:");
   scanf("%s",t->name);
   printf("Enter
   Branch:");
   scanf("%s",t->branch);
   printf("Enter Sem:");
   scanf("%d",&t->sem);
   printf("Enter Phno:");
   scanf("%s",t->phno);
   t-       >next=N
   ULL; if(!r)
      f=r=t
   ; else
   {
      if(ch)
      {
         r->next=t;
         r=t;
      }
      else
      {
         t->next=f
         ; f=t;
      }
   }
```

```
}
void del(int ch)
{
   if(!f)
      printf("\nList Empty");
```

```
    else
    {
       struct stud
       *t1; if(f==r)
       {
          t1=f;
          f=r=NULL;
       }
       else if(ch)
       {
          t1=r;
          for(t=f;t->next!=r;t=t->next)
             r=t;
          r->next=NULL;
       }
       else
       {
          t1=f;
          f=f->next;
       }
       printf("\nElement deleted is:\n");
       printf("USN:%s\nName:%s\nBranch:%s\nSem:%d\nPhno:%s\n",t1->usn,t1->name
       ,
                t1->branch,t1->sem,t1->phno);
       free(t1);
    }
}
void disp()
{
    if(!f)
       printf("\nList
    Empty!!!"); else
       printf("\nList elements
    are:\n"); for(t=f;t;t=t->next)
       printf("\nUSN:%s\nName:%s\nBranch:%s\nSem:%d\nPhno:%s\n",t->usn,t->name,
              t->branch,t->sem,t->phno);
}
void main()
{
    int    ch,n,i;
    printf("\n.....Menu
```

```
,\n");
printf("1.Create\n");
printf("2.Display\n");
printf("3.Insert at end\n");
printf("4.Delete at
end\n"); printf("5.Insert at
beg\n");
```

```
    printf("6.Delete at beg\n");
    printf("7.Exit\n")
    ; while(1)
    {
       printf("\nEnter
       choice:");
       scanf("%d",&ch);
       switch(ch)
       {
          case 1: printf("\nEnter no. of
                  nodes:"); scanf("%d",&n);
                  for(i=0;i<n;i++)
                     ins(0)
                  ; break;
          case 2:disp();break;
          case 3:ins(1);break;
          case 4:del(1);break;
          case 5:ins(0);break;
          case 6:del(0);break;
          case 7:exit(0);
          default:printf("\nInvalid choice!!!!");
       }
    }

}
```

Output

.....Menu. ,

1. Create

2.Display

3.Insert at end

4.Delete at

end 5.Insert at

beg 6.Delete

at beg 7.Exit

Enter choice:1

Enter no. of

nodes:1 Enter

USN:23 Enter

Name:aaa Enter

Branch:cse Enter

Sem:3

Enter

Phno:9823456789

Enter choice:2

List elements are:

USN:23

Name:aaa

Branch:cs

e Sem:3

Phno:9823456789

Enter choice:3

Enter USN:34

Enter Name:bbb

Enter

Branch:cse

Enter Sem:3

Enter

Phno:8792346758

Enter choice:2

List elements are:

USN:23
Name:a
aa
Branch:
cse
Sem:3
Phno:9823456789

USN:34
Name:bbb
Branch:cs
e Sem:3
Phno:8792346758

Enter choice:4
Element deleted is:
USN:34
Name:bbb
Branch:cs
e Sem:3
Phno:8792346758
Enter choice:6
Element deleted is:
USN:23
Name:aaa
Branch:cs
e Sem:3
Phno:9823456789

Enter choice:5
Enter USN:23
Enter Name:aaa
Enter
Branch:cse
Enter Sem:3
Enter
Phno:8792356789
Enter choice:2

USN:23

Name:a

aa

Branch:

cse

Sem:3

Phno:9823456789

List elements are:

USN:23

Name:aaa
Branch:cs
e Sem:3
Phno:8792356789

Program outcome :

- Implement Singly Linked List.
- Implement insertion at the front and end of SLL.
- Implement deletion at the front and end of SLL.
- Identify the applications of SLL.
- Familiarized how SLL can be used as both stack and queue.

Viva Questions :

- What is a Linked List and what are its types? What is a node?
- What are the parts of a linked list? What are the advantages of linked list?
- Mention what is traversal in linked lists?

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

Program objective:

- Understand the Doubly Linked List (DLL) data structures.
- Understand the methodology to insert and delete the element at the frontof DLL.
- Understand the methodologyto insert and delete the element at the endof DLL.
- Get the knowledge of how DLL and be used as double ended queue.

Algorithm

Insertion at front end of list.

Step 1: Allocate memory for temp node and assign values to node
Step2: if list is empty, temp is attached to list directly
head=nu
ll
returnte
mp
if list is not empty
temp->rlink=head
head->llink=temp
return head

Insertion at rear end of list.

Step1: Read node information and allocate memory for temp node

Step2: traverse the cur node upto to end of list then attach node cur to temp cur->rlink=temp;
temp->llink=cur

Step 3:return starting address of list

return head;

Delete from front end of list.

Step 1: check if list has only one
node head=NULL;
return null;
if list is
empty
head->rlink
=NULL
return
NULL;
if list has o0nly one node
Step 2:otherwise first node address is shifted to next node
cur=head
head=he
ad->rlin
k
free(cur)
Step 3: return starting address of list
return head

Delete node from rear end

Step 1: two pointers requires one is cur and prev
Cur is one which points, node to be deleted.

Step 2: Traverse the cur node upto end of list before updating current pointer save the
Address to prev pointer.

```
While(cur->rlink!=null)
{
prev=c
ur;
cur=cu
r->rlin
k;
}
prev->rlink
=null;
```

cur->llink=
null;
free(cur);

Step3: return starting address of the list

THEORY

- 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 and to 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 of the previous node during traversal or no need to traverse the list to find the previous node, so that its link can be modified.

PROGRAM:

```
#include<string.h
> int count=0;
struct node
{
struct node
*prev; int
ssn,phno;
float sal;
char name[20],dept[10],desg[20];
struct node *next;
}*h,*temp,*temp1,*temp2,*temp4
; void create()
{
int ssn,phno;
float sal;
char name[20],dept[10],desg[20];
 temp =(struct node *)malloc(sizeof(struct
 node)); temp->prev = NULL;
 temp->next = NULL;
 printf("\n Enter ssn,name,department, designation, salary and phno of employee :
 "); scanf("%d %s %s %s %f %d", &ssn, name,dept,desg,&sal, &phno);
 temp->ssn = ssn;
 strcpy(temp->name,name)
 ; strcpy(temp->dept,dept);
 strcpy(temp->desg,desg);
 temp->sal = sal;
 temp->phno = phno;
 count++;
}
void insertbeg()
{
if (h == NULL)
 {
create();
 h =
 temp;
 temp1 = h;
```

```
}
else
{
create();
temp->next = h;
h->prev =
temp;
```

```
 h = temp;
 }
}
void insertend()
{
if(h==NULL
)
 {
 create();
 h =
 temp;
 temp1 = h;
 }
else
 {
 create();
 temp1->next          =
 temp;  temp->prev  =
 temp1;     temp1     =
 temp;
 }
}
void displaybeg()
{
 temp2 =h;
if(temp2 == NULL)
 {
 printf("List empty to display \n");
 return;
 }
 printf("\n Linked list elements from begining : \n");
while (temp2!= NULL)
 {
 printf("%d %s %s %s %f %d\n", temp2->ssn,
 temp2->name,temp2->dept, temp2->desg,temp2->sal, temp2->phno );
 temp2 = temp2->next;
}
 printf(" No of employees = %d ", count);
}
```

```
int deleteend()
{
struct node
 *temp; temp=h;
if(temp->next==NULL)
 {
 free(temp)
 ;
 h=NULL;
 return 0;
```

```
 }
else
 {

 temp2=temp1->prev;
 temp2->next=NULL
 ;
 printf("%d %s %s %s %f %d\n", temp1->ssn,
 temp1->name,temp1->dept, temp1->desg,temp1->sal, temp1->phno );
 free(temp1);
 }
 count--
; return
0;
}
int deletebeg()
{
struct node
 *temp; temp=h;
if(temp->next==NULL)
 {
 free(temp)
 ;
 h=NULL;
 }
else
 {
 h=h->next;
 printf("%d %s %s %s %f %d", temp->ssn, temp->name,temp->dept,
 temp->desg,temp->sal, temp->phno );
 free(temp);
 }
 count--
; return
0;
}
void main()
{
int ch,n,i;
 h=NULL
```

```
;
temp = temp1 = NULL;
printf("_____MENU_____
\n"); printf("\n 1 - create a DLL of n emp");
printf("\n 2 - Display from beginning");
printf("\n 3 - Insert at end");
printf("\n 4 - delete at
end"); printf("\n 5 - Insert at
beg"); printf("\n 6 - delete
at beg"); printf("\n 7 -
exit\n");
printf("_____\n");
```

```
while (1)
 {
 printf("\n Enter choice :
 "); scanf("%d", &ch);
 switch (ch)
 {
 case 1:
 printf("\n Enter no of employees : ");
 scanf("%d", &n);
 for(i=0;i<n;i++
 ) insertend();
 break;
 case 2:
 displaybeg();
 break;
 case 3:
 insertend()
 ; break;
 case 4:
 deleteend()
 ; break;
 case 5:
 insertbeg()
 ; break;
 case 6:
 deletebeg()
 ; break;
 case 7:
 exit(0);
 default
 :
printf("wrong choice\n");
 }
 }
}
```

Output

._____MENU_____

1 - create a DLL of n emp
2 - Display from
beginning 3 - Insert at end
4 - delete at
end 5 - Insert at
beg 6 - delete
at beg 7 - exit

Enter choice : 1

Enter no of employees : 2

Enter ssn,name,department, designation, salary and phno of employee :
123 ASG
CSE
Associate
20000
675432

Enter ssn,name,department, designation, salary and phno of employee :
546 PPT
CSE
Professor
40000
9887654

Enter choice : 2

Linked list elements from begining :
123 ASG CSE Associate 20000.000000 675432
546 PPT CSE Professor 40000.000000 9887654
No of employees = 2
Enter choice : 3

Enter ssn,name,department, designation, salary and phno of employee :
789 TTR
ECE

Associate
25000

988765

Enter choice : 2

Linked list elements from begining :

123 ASG CSE Associate 20000.000000 675432

546 PPT CSE Professor 40000.000000 9887654

789 TTR ECE Associate 25000.000000 988765

No of employees = 3

Enter choice : 4

789 TTR ECE Associate 25000.000000 988765

Enter choice : 2

Linked list elements from begining :

123 ASG CSE Associate 20000.000000 675432

546 PPT CSE Professor 40000.000000 9887654

No of employees = 2

Enter choice : 7

-

Program outcome:

- Implement Doubly Linked List.
- Implement insertion at the front and end of DLL.
- Implement deletion at the front and end of DLL.
- Identify the applications of DLL.
- Familiarized how DLL can be used as double ended queue.

Viva Questions:

- What are doubly linked lists?
- What is the difference between singly and doubly linked lists?
- What are the advantages of double linked list over single linked list?

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)

Program objective: .

- Understand the working of Singly Circular Linked List (SCLL).
- Understand the use of header nodes.
- Understand the methodology to evaluate polynomial using SCLL.
- Understand the methodology to add two polynomial using SCLL.

Algorithm:

Evaluate a Polynomial

Step1: allocate memory for newly created node assign values to that node
Step 2: attach newly created node to list in circular fashion.
Step3: Evaluate each node information up to header node

Addition of two Polynomial

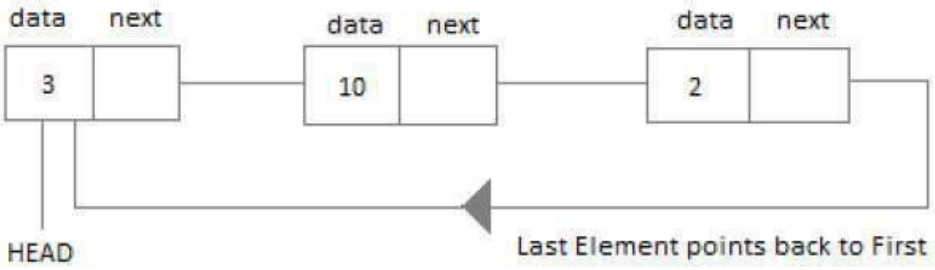
Step1: Read exponent values and co-efficient values for each node
Step2: newly created node are attached to polynomials (p1, p2, p3)
Step3: Addition/Evaluation of list is performed
Step 5: Result is displayed

THEORY

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.

Circular Linked List is a variation of Linked list in which the first element points to the last element and the last element points to the first element. Both Singly Linked List and Doubly Linked List can be made into a circular linked list.



PROGRAM:

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<math.h>
typedef struct
node
{
int expo,coef;
struct node *next;
}node;
/*FUNCTION PROTOTYPE*/
node * insert(node
*,int,int); node * create();
node * add(node *p1,node
*p2); int eval(node *p1);
void display(node *head);
node *insert(node*head,int expo1,int coef1)
{
 node *p,*q;
 p=(node
 *)malloc(sizeof(node));
 p->expo=expo1;
 p->coef=coef1;
p->next=NULL;
if(head==NULL
)
 {
 head=p;
 head->next=head
 ; return(head);
 }

if(expo1>head->expo)
 {
 p-      >next=head->
```

```
next; head->next=p;
head=p;
return(head);
}
if(expo1==head->expo)
{
head->coef=head->coef+coef1;
```

```
 q=head;
while(q->next!=head&&expo1>=q->next->expo
 ) q=q->next;
if(p->expo==q->expo)
 q-   >coef=q->coef+coe
f1; else
 {
 p->next=q->next
 ; q->next=p;
 }
return(head);
}
node *create()
{
int n,i,expo1,coef1;
 node
 *head=NULL;
 printf("\n\nEnter no of terms of
 polynomial==>"); scanf("%d",&n);
for(i=0;i<n;i++)
 {
 printf("\n\nEnter coef &
 expo==>");
 scanf("%d%d",&coef1,&expo1);
 head=insert(head,expo1,coef1);
 }
return(head);
}
node *add(node *p1,node *p2)
{
 node *p;
 node *head=NULL;
 printf("\n\n\nAddition of
 polynomial==>"); p=p1->next;
do
 {
 head=insert(head,p->expo,p->coef)
```

```
 ; p=p->next;
 }while(p!=p1->next)
 ; p=p2->next;
do
 {
head=insert(head,p->expo,p->coef)
 ; p=p->next;
 }while(p!=p2->next);
```

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

do
 {
 if(n-1)
 {
 printf("%dx^(%d) +
 ",p->coef,p->expo); p=p->next;
 }
 else
 {
printf("
 %dx^(%d)",p->coef,p->expo);
 p=p->next;
 }
```

```
 n--;
 } while(p!=head->next);
}
void main()
{
int a,x,ch;
```

```
node *p1,*p2,*p3;
p1=p2=p3=NULL;
while(1)
{
printf("\n\t----------<< MENU >>                    ");
printf("\n\tPolynomial Operations :");
printf(" 1.Add");
printf("\n\t\t\t\t2.Evaluate");
printf("\n\t\t\t\t3.Exit");
printf("\n\t_____
"); printf("\n\n\n\tEnter your choice==>");
scanf("%d",&ch);
switch(ch)
{
case 1 :
p1=create();
display(p1);
p2=create();
display(p2);
p3=add(p1,p2);
display(p3);
break;
case 2 :
p1=create()
;
display(p1);
a=eval(p1);
printf("\n\nValue of polynomial=%d",a);
break;
case 3 :
exit(0);
break;
default :
printf("\n\n\t invalid choice");
break;
}
}
}
```

Output:

```
_____<< MENU >>_____
    Polynomial Operations : 1.Add
                            2.  Eva
    ----------              ----------
                            luate
                            3.Exit

    Enter your choice==>1
Enter no of terms of
polynomial==>3 Enter coef &
expo==>2 4
Enter coef & expo==>3 2
Enter coef & expo==>1 0
    The polynomial is==>1x^(0) + 3x^(2) + 2x^(4)
Enter no of terms of polynomial==>2
Enter coef & expo==>2 3
Enter coef & expo==>2 0

    The polynomial is==>2x^(0) + 2x^(3)

Addition of polynomial==>

    The polynomial is==>3x^(0) + 3x^(2) + 2x^(3) + 2x^(4)
    _____<< MENU >>_____
    Polynomial Operations : 1.Add
    ----------              ----------
                            2.Evaluate
                            3.Exit

    Enter your choice==>2

Enter no of terms of
polynomial==>3 Enter coef &
expo==>3 4
Enter coef & expo==>2 3
Enter coef & expo==>1 0
    The polynomial is==>1x^(0) + 2x^(3) + 3x^(4)


Enter the value of x=2
```

Value of polynomial=65

<< MENU >>

Polynomial Operations : 1.Add

2.Evaluate

3.Exit

Enter your choice==>3

Program outcome :

- Implement Singly Circular Linked List (SCLL) using header node.
- Identify the application ofSCLL.
- Familiarized withthe methodology ofpolynomial evaluation andpolynomial addition using SCLL.

Viva Questions:

- What is circular linked list.?
- What are Advantages and Disadvantages of Circular Linked List?

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**

Program objective:

- Understand the concept of Binary Search Tree (BST).
- Understand the different traversal method on BST.
- Get to know the methodology of searching a key element in BST.
- Understand the methodology of deleting an element from BST.

Algorithm:

Preorder Traversal

Step 1: Display root information

Step2: Traverse left sub tree in preorder
Step 3: Traverse right sub tree in preorder

In order Traversal

Step 1: Traverse the left sub tree in order
Step 2: Display root information
Step3: Traverse right sub tree in order

Post order Traversal

Step 1: traverse the left sub tree in post order
Step 2: traverse the right sub tree in post order
Step 3: Display root information

‘

THEORY

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 equalto its parentnode‘s V key.
- The right sub-tree of a node has key greater than or equal to its parentnode'skey.

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)

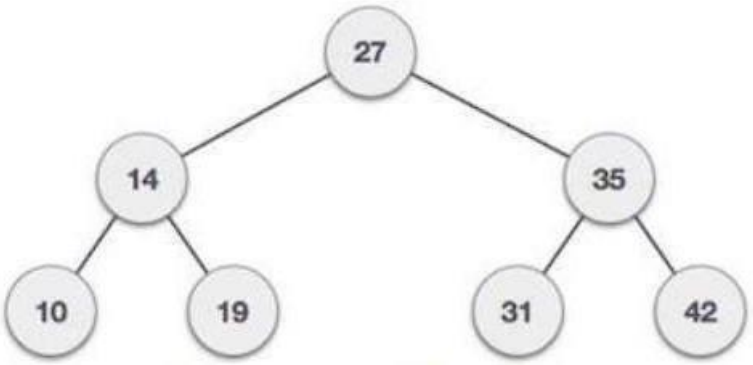


Fig: An example of BST

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 apreordermanner.
- **Inorder Traversal** − traverse a tree in an inordermanner.
- **Postorder Traversal** − traverse a tree in a postorder manner.

PROGRAM:

```
#include  <stdio.h>
#include
<stdlib.h>          int
flag=0;
typedef struct BST
{
int data;
struct BST *lchild,*rchild;
} node;
/*FUNCTION PROTOTYPE*/
void insert(node *, node
*); void inorder(node *);
void preorder(node *);
void postorder(node *);
node *search(node *, int, node
**); void main()
{
int choice;
int       ans
=1;       int
key;
 node      *new_node,      *root,      *tmp,
 *parent; node *get_node();
 root = NULL;
 printf("\nProgram For Binary Search Tree ");
do
 {
 printf("\n1.Create");
 printf("\n2.Search");
 printf("\n3.Recursive Traversals");
 printf("\n4.Exit");
 printf("\nEnter your choice
 :"); scanf("%d", &choice);
 switch (choice)
 {
 case 1:
 do
 {
```

```
 new_node = get_node();
 printf("\nEnter The Element ");
 scanf("%d", &new_node->data);
if (root == NULL) /* Tree is not Created */
 root = new_node;
 else
```

```
insert(root, new_node);
printf("\nWant To enter More Elements?(1/0)");
scanf("%d",&ans);
} while
(ans); break;
case 2:
printf("\nEnter Element to be searched
:"); scanf("%d", &key);
tmp = search(root, key, &parent);
if(flag==1)
{
printf("\nParent of node %d is %d", tmp->data, parent->data);
}
else
{
printf("\n The %d Element is not Present",key);
}
flag=0
;
break;
case 3:
if (root == NULL)
printf("Tree Is Not Created");
else
{
printf("\nThe Inorder display
:"); inorder(root);
printf("\nThe Preorder display :
"); preorder(root);
printf("\nThe Postorder display :
"); postorder(root);
}
break;
}
}
while (choice != 4);
}
/*Get new Node */
node *get_node()
{
node *temp;
```

```
temp = (node *)
 malloc(sizeof(node)); temp->lchild =
 NULL;
 temp->rchild =
NULL; return temp;
}
```

```
/*This function is for creating a binary search tree */
void insert(node *root, node *new_node)
{
if (new_node->data < root->data)
 {
 if(root->lchild==NULL
 )
 root->lchild=new_node;
 else
 insert(root->lchild, new_node);
 }
if (new_node->data > root->data)
 {
 if (root->rchild ==
 NULL) root->rchild =
 new_node; else
 insert(root->rchild, new_node);
 }
}
/*This function is for searching the node from binary Search Tree*/
node *search(node *root, int key, node **parent)
{
 node
 *temp;
 temp = root;
while (temp != NULL)
 {
 if (temp->data == key)
 {
 printf("\nThe %d Element is Present", temp->data);
 flag=1;
 return temp;
 }
 *parent = temp;
 if (temp->data >
 key) temp =
 temp->lchild; else
 temp = temp->rchild;
 }
return NULL;
```

```
}
/*This function displays the tree in inorder fashion
*/ void inorder(node *temp)
{
if (temp != NULL)
 {
 inorder(temp->lchild);
```

```
 printf("%d\t",
 temp->data);
 inorder(temp->rchild);
 }
}
/*This function displays the tree in preorder fashion */
void preorder(node *temp)
{
if (temp != NULL)
 {
 printf("%d\t",
 temp->data);
 preorder(temp->lchild);
 preorder(temp->rchild);
 }
}
/*This function displays the tree in postorder fashion */
void postorder(node *temp)
{
if (temp != NULL)
 {
 postorder(temp->lchild);
 postorder(temp->rchild);
 printf("%d\t",
 temp->data);
 }
}
```

Output

Program For Binary Search Tree
1.Create
2.Search
3. Recursive Traversals 4.Exit
Enter your choice :1

Enter The Element 6

Want To enter More Elements?(1/0)1

Enter The Element 9

Want To enter More Elements?(1/0)1

Enter The Element 5

Want To enter More Elements?(1/0)1

Enter The Element 2

Want To enter More Elements?(1/0)1

Enter The Element 8

Want To enter More Elements?(1/0)1

Enter The Element 15

Want To enter More Elements?(1/0)1

Enter The Element 24

Want To enter More Elements?(1/0)1

Enter The Element 14

Want To enter More Elements?(1/0)1

Enter The Element 7

Want To enter More Elements?(1/0)1

Enter The Element 8

Want To enter More Elements?(1/0)1

Enter The Element 5

Want To enter More Elements?(1/0)1

Enter The Element 2

Want To enter More Elements?(1/0)0

1.Create
2.Searc
h
3. Recursive
Traversals 4.Exit
Enter your choice :2

Enter Element to be searched :8

The 8 Element is Present
Parent of node 8 is 9
1.Create
2.Search
3. Recursive
Traversals 4.Exit
Enter your choice :3

The Inorder display :2 5 6 7 8 9 14 15 24

The Preorder display : 6 5 2 9 8 7 15 14 24

The Postorder display : 2 5 7 8 14 24 15 9 6
1.Create
2.Search
3. Recursive
Traversals 4.Exit
Enter your choice :4

Program outcome:

- Implement Binary Search Tree (BST).
- Implement the different traversal methodology on BST.
- Familiarized with the methodology to search a key element in BST.
- Implement the methodology to delete an element from BST.
- Identify the applications of BST

Viva Questions:

- What are binary trees?
- Explain Binary Search Tree
- How to check if a given Binary Tree is BST or not?
- What is the minimum number of nodes that a binary tree can have?
- What are the different types of traversing?
- Define pre-order traversal?
- Define post-order traversal?
- Define in -order traversal?

PROGRAM 11

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

Program objective:

- Understand the concept of trees and adjacency matrix.
- Understand the concept of connected graph.
- Understand the Breath First Search (BFS) and Depth First Search(DFS)traversal methodologies.

Algorithm:

Step 1: Initialize front,rear,visit and number of nodes

Step 2: Read adjacency matrix for graph

Step 3: select source vertex from graph i.e v

Step 4: source node is added into queue and cover all the nodes (adjacent) to v.

Once it is covered adjacent/traversed mark as visited.

Step 5: Read next vertex from queue and cover all the nodes .if it is not visited,

visit the nodes.

Step 6: Repeat the process 3-5 until all nodes are covered in queue

THEORY

BFS first visits all the vertices that are adjacent to a starting vertex. Every time it adds the adjacent vertex to a queue array q. On each successive iteration of the algorithm, the next vertex on the queue is examined to see if there are any unvisited vertices adjacent to it which can be added to the queue. Whenever a new vertex is taken from the queue, it is marked as a visited node in the visited array.

Applications of BFS:

- To check connectivity of a graph (number of times queue becomes emptytellsthe number of components in thegraph)
- To check if a graph is acyclic. (no cross edges indicates no cycle)
- To find minimum edge path in a graph

Depth first search is a graph algorithm required for processing vertices or edges of a graph in a systematic fashion. Depth first search starts visiting vertices of a graph at an arbitrary vertex by marking it as having been visited. On each iteration, the algorithm proceedsto an unvisited vertex that is adjacent to one it is currently in.

The algorithm backs up one edge to the vertex it came from and tries to continue visiting unvisited vertices from there. The algorithm eventually halts after backing up to starting vertex, with the latter being dead end. By then, all vertices in the same connected component as the starting vertex have been visited. If unvisited vertices still remain, the depth first search must be restarted at any one of them.

Here we use a **STACK** to trace the depth first search. We push a vertex onto the stack when the vertex is reached for the first time, and we pop a vertex off the **stack** when it becomes a dead end.

Applications of DFS:

- The two orderings are advantageous for various applications liketopologicalsorting, etc.
- To check connectivity ofa graph (number oftimes stack becomes emptytells the number of components in thegraph)
- To check if a graph is acyclic. (no back edges indicates no cycle)
- To find articulation point in a graph

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
int a[20][20],q[20],visited[20],reach[10],n,i,j,f=0,r= -1,count=0;
void bfs(int v)
{
for(i=1;i<=n;i++)
 if(a[v][i] && !visited[i])
 q[++r]=i;
if(f<=r)
 {
 visited[q[f]]=1;
 bfs(q[f++]);
 }
}
void dfs(int v)
{
int i;
 reach[v]=1
 ;
for(i=1;i<=n;i++)
 {
 if(a[v][i] && !reach[i])
 {
 printf("\n %d->%d",v,i);
 count++;
 dfs(i);
 }
 }
}
void main()
{
int v, choice;
 printf("\n Enter the number of
 vertices:"); scanf("%d",&n);
for(i=1;i<=n;i++)
 {
 q[i]=0
```

```
;
 visited[i]=0;
 }
for(i=1;i<=n-1;i++
 ) reach[i]=0;
printf("\n Enter graph data in matrix
form:\n"); for(i=1;i<=n;i++)
 for(j=1;j<=n;j++)
```

```
 scanf("%d",&a[i][j]);
printf("1.BFS\n 2.DFS\n 3.Exit\n");
scanf("%d",&choice);
switch(choice)
 {
 case 1:
 printf("\n Enter the starting vertex:");
 scanf("%d",&v);
 bfs(v);
 if((v<1)||(v>n))
 {
 printf("\n Bfs is not possible");
 }
 else
 {
 printf("\n The nodes which are reachable from %d:\n",v);
 for(i=1;i<=n;i++)
 if(visited[i])
 printf("%d\t",i)
 ;
 }
 break;
 case 2:
 dfs(1);
 if(count==n-1
 )
 printf("\n Graph is connected");
 else
 printf("\n Graph is not
 connected"); break;
 case 3:
 exit(0)
 ;
 }
}
```

OUTPUT:

Enter the number of vertices:4

Enter graph data in matrix
form: 0 2 0 2
2 0 0 3

1 0 0 4

2 3 4 0

1.BFS
2.DFS

3. E
xit 1

Enter the starting vertex:1

The nodes which are reachable from
1: 1 2 3 4

Program outcomes:

- Create graph using adjacency matrix.
- Implement Breadth First Search (BFS) and Depth First Search(DFS).
- Familiarized with connected graph.
- Identify the applications of graphs.

Viva Questions:

- What is a graph?
- What is a tree?
- What is BFS and DFS?
- Which data structures are used for BFS and DFS of a graph?

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.

Program objective:

- Understand what is hashing and hashing function.
- Understand the concept of linear probing.
- Understand the concept ofcollision detectionand avoidance usinglinear probing.

Algorithm:

Step 1: Start

Step 2: Initialize all memory locations with some values to identity as space a[i]=-1

Step 3: Read Employee key value .calculate hash key using remainder method hk<-key%100

Step 4: Inserting Employee record using key

Inserting hash dull function

If(count=m)

If space is available for that key If(H[k]==-1)

H[hk] <-key

If collision occurs, it can be solved using linear probing method.

Checking free space from key to end

for(i=hk+1;i<m;i++)

Checking free space from beginning to key value. for(i=0;i<hk&& flag==0;i++)

Step 5: Display all memory location with index and employee key

```
#include  <stdio.h>
#include
<stdlib.h>  #define
MAX 100
/*FUNCTION PROTOTYPE */
int create(int);
void linear_prob(int[], int, int);
void display (int[]);
void main()
{
int a[MAX],num,key,i;
int ans=1;
 printf(" collision handling by linear probing : \n");
for (i=0;i<MAX;i++)
 {
 a[i] = -1;
 }
do
 {
 printf("\n Enter the
 data"); scanf("%4d",
 &num); key=create(num);
 linear_prob(a,key,num);
 printf("\n Do you wish to continue ? (1/0) ");
 scanf("%d",&ans);
 }while(ans)
 ; display(a);
}
int create(int num)
{
int key;
key=num%100;
return key;
}
void linear_prob(int a[MAX], int key, int num)
{
int flag, i, count=0;
```

```
flag=0;
if(a[key]== -1)
{
 a[key] = num;
 }
else
 {
```

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

```
int i,choice;
 printf("1.Display ALL\n 2.Filtered Display\n");
 scanf("%d",&choice);
if(choice==1)
 {
 printf("\n the hash table is\n");
 for(i=0; i<MAX; i++)
```

```
 printf("\n %d %d ", i, a[i]);
 }
else
 {
 printf("\n the hash table
 is\n"); for(i=0; i<MAX; i++)
 if(a[i]!=-1)
 {
 printf("\n %d %d ", i,
 a[i]); continue;
 }
 }
}
```

Output

Collision handling by linear probing :

Enter the data: 1234

Do you wish to continue? (1/0) 1

Enter the data: 2345

Do you wish to continue? (1/0) 1

Enter the data: 1234

Collision Detected...!!!

Collision avoided successfully using LINEAR

PROBING Do you wish to continue? (1/0) 1

Enter the data: 2345

Collision Detected...!!!

Collision avoided successfully using LINEAR

PROBING Do you wish to continue? (1/0) 0

1. Display ALL

2. Filtered

Display 2

The hash table

is 34 1234

35 1234

45 2345

46 2345

Program outcome:

- Implement hashing function.
- Implement linear probing.
- Familiarized the concept ofcollusion detectionand avoidance anddetection using linear probing.
- Identify the application of hashing and linear probing.

Viva Questions:

- What is Hashing?
- What is Linear Probing?