DATA STRUCTURES LABORATORY
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 – 1			

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

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: 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). 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 Support the program with functions for each of the above operations. Don't use Built-in functions.
3.	Design 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 alphanumeric operands.

5.	Design 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 *frontinsertion*. b. Display the status of SLL and count the number of nodes in it c. Perform Insertion / Deletion at EndofSLL d. Perform Insertion / Deletion at Front of SLL(Demonstration ofstack) 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 *endinsertion*. b. Display the status of DLL and count the number of nodes init c. Perform Insertion and Deletion at End ofDLL d. Perform Insertion and Deletion at Front ofDLL e. Demonstrate how this DLL can be used as Double EndedQueue. f. 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. 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 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
  - o For laboratories having only one part: Students are allowed to pick one experiment from the lot with equal opportunity.
  - o For laboratories having PART A and PART B: Students are allowed to pick one experiment from PART A and one experiment from PART B, with equal opportunity.
- Change of experiment is allowed only once and marks allotted for procedure to be made zero of the changed part only.
- Marks Distribution *(Need to change in accordance with university regulations)*
  - a) For laboratories having only one part – Procedure + Execution + Viva-Voce: 15+70+15 = 100 Marks
  - b)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

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 classifiedas:

- 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. Rather its value can be accessed directly by referencing a variable).

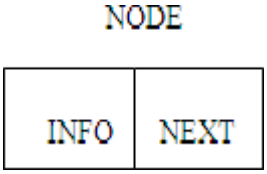

Fig-1 Linked List

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

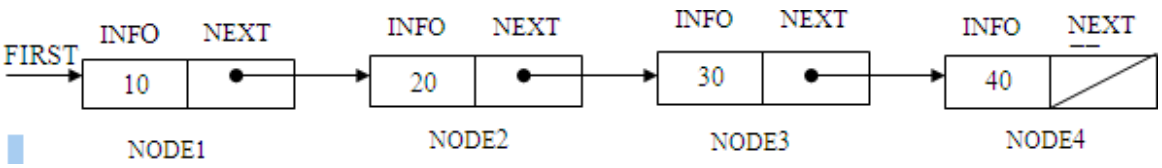

Fig-2 Linked List

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

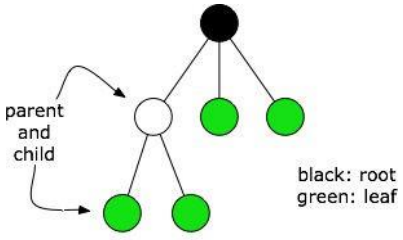

Figure: tree data structure

Fig-3 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 (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.**

Program objective:

- Understand the working of array data structures.
- Understand the use of functions to implement each array operation.
- Understand what is dynamic memory allocation

THEORY:

Array is a collection of elements of the same type. In this program we need to use functions for various operations
Create (): Create an array for the size given by the user Display
(): Display the elements of the array
Insert (): Insert an element at the position given by the user
Delete (): Delete an element from the position specified by the user Exit
(): Terminate

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 arrayas a collection of variables of the sametype.

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 arrayName [arraySize];

This is called a *single-dimensional* array. The **array Size** must be an integer constant greater than zero and **type** can be any valid C data type.

For example, to declare a 10-element array called **balance** of type double, use this statement –
double balance [10];

Here *balance* is a variable array which is sufficient to hold up to 10 double numbers. Initializing Arrays

You can initialize an array in C either one by one or using a single statement as follows –
double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};

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

PROGRAM:

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

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

// Function to create the calendar
struct Day* createCalendar()
{
   struct Day* calendar = (struct Day*)malloc(7 * sizeof(struct Day));

   for (int i = 0; i < 7; i++)
  {
        calendar[i].dayName = (char*)malloc(20 * sizeof(char)); // Assuming a
maximum of 20  characters for day name
        calendar[i].activity = (char*)malloc(100 * sizeof(char)); // Assuming a
maximum of 100 characters for activity description
   }

   return calendar;
}
```

```
// Function to read data from the keyboard
void readCalendarData(struct Day* calendar)
{
	for (int i = 0; i < 7; i++)
	{
		printf("Enter the day name for Day %d: ", i + 1);
		scanf("%s", calendar[i].dayName);

		printf("Enter the date for Day %d: ", i + 1);
		scanf("%d", &calendar[i].date);

		 printf("Enter the activity for Day %d: ", i + 1);
		 scanf(" %[^\n]s", calendar[i].activity);
	}
}

// Function to display the calendar
void displayCalendar(struct Day* calendar)
{
	printf("Weekly Activity Report:\n\n");
	for (int i = 0; i < 7; i++)
	{
		printf("Day %d: %s\n", i + 1, calendar[i].dayName);
		printf("Date: %d\n", calendar[i].date);
		printf("Activity: %s\n", calendar[i].activity);
		printf("\n");
	}
}

int main()
{
	struct Day* calendar = createCalendar();

	readCalendarData(calendar);
	displayCalendar(calendar);

	// Free memory
	for (int i = 0; i < 7; i++)
	{
		free(calendar[i].dayName);
		free(calendar[i].activity);
	}
	free(calendar);
```

```
    return 0;
}
```

Output

```
Enter the day name for Day 1: Monday
Enter the date for Day 1: 12
Enter the activity for Day 1: NSS
Enter the day name for Day 2: Tuesday
Enter the date for Day 2: 13
Enter the activity for Day 2: Project work
Enter the day name for Day 3: Wednesday
Enter the date for Day 3: 14
Enter the activity for Day 3: Assignment
Enter the day name for Day 4: Thursday
Enter the date for Day 4: 15
Enter the activity for Day 4: Seminar
Enter the day name for Day 5: Friday
Enter the date for Day 5: 16
Enter the activity for Day 5: Council Meeting
Enter the day name for Day 6: Saturday
Enter the date for Day 6: 17
Enter the activity for Day 6: Project Report
Enter the day name for Day 7: Sunday
Enter the date for Day 7: 18
Enter the activity for Day 7: Holiday

Weekly Activity Report:

Day 1: Monday
Date: 12
Activity: NSS

Day 2: Tuesday
Date: 13
Activity: Project work

Day 3: Wednesday
Date: 14
Activity: Assignment

Day 4: Thursday
Date: 15
Activity: Seminar
```

Day 5: Friday
Date: 16
Activity: Council Meeting

Day 6: Saturday
Date: 17
Activity: Project Report

Day 7: Sunday
Date: 18
Activity: Holiday

Program outcome:

- Implement the arrays in C program
- Implement dynamically creation and allocation of arrays
- Identify the different applications where arrays can be used.
- Familiarized with the usage of structures and functions in program.

Viva Questions:

- What is an array?
- What is a structure ?
- How to access elements of array?
- Can you change size of array once created?
- What is dynamic memory management?
- How An array elements are always stored in memory locations.?

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 theimplementationtechniqueof the 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 read();
void match();
char    STR[100],PAT[100],REP[100],ANS[100];
int c,i,j,k,m,flag=0;
main()
{
read();
match();
}
void read()
{
printf("enter the main string STR:");
gets(STR);
printf("enter pattern string PAT:");
gets(PAT);
printf("enter replace string REP:");
gets(REP);
}
void match()
{
c=i=j=k=m=0;
while(STR[c]!='\0')
{
if(STR[m]==PAT[i])
{
i++;m++;
flag=1;
if(PAT[i]=='\0')
{

for(k=0;REP[k]!='\0';k++,j++)

ANS[j]=REP[k];
i=0;
```

```
    c=m;

    }
    }
    else
    {
    ANS[j]=STR[c];
    j++;c++;
    m=c;
    i=0;
    }
    }
    if(flag==0)
    printf("pattern not found");
    else
    {
    ANS[j]='\0';
    printf("resultant string is %s",ANS);
    }
}
```

Output 1

Enter the MAIN string:
atme college of engg
Enter a PATTERN string:
engg
Enter a REPLACE string:
engineering
The RESULTANT string is: atme college of engineering

Output 2

Enter the MAIN string:
atme college of engg
Enter a PATTERN string:
for
Enter a REPLACE string:
if
Pattern doesn't found!!!

Output 3

Enter the MAIN string:
This is Data Structure lab
Enter a PATTERN string:
Data Structure
Enter a REPLACE string:
Data structure with C
The RESULTANT string is: This is Data structure with C lab

Program outcomes:

- Implement string matchingand string replacement algorithmwithout using built-in library functions.
- Apply the knowledge of array usage to implement string functions.
- Identify different applications of string matching andstringreplacement.

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 Cprogramming?

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 ontoStack

b Pop an Element fromStack

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 onstacks i.e., push, pop and implement the same.
- 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 −

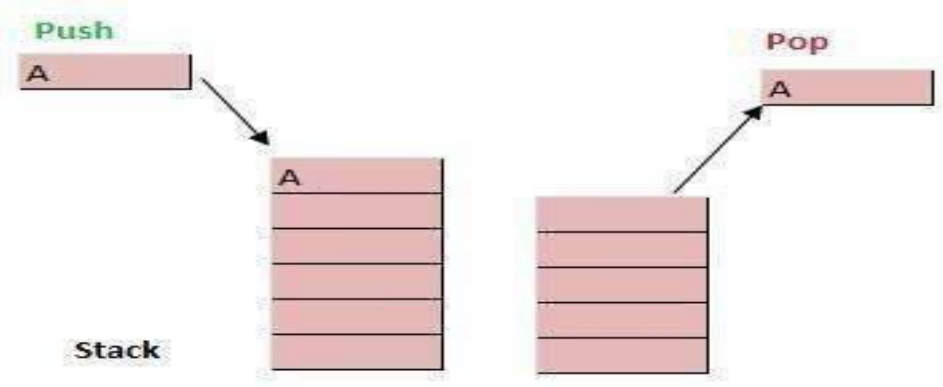

Fig4-Example of Stack

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 MAX 4
int stack[MAX],top=-1,item;
void push();
void pop();
void palindrome();
void display();
void main()
{
int choice;
while(1)
{
Printf("------- STACK OPERATIONS ------\n");
printf("1.push\n 2.pop\n 3.palindrome\n 4.display\n 5.exit\n");
printf("enter choice");
scanf("%d",&choice);
switch(choice)
{
case 1:push();
      break;
case 2:pop();
      break;
case 3:palindrome();
      break;
case 4:display();
      break;
case 5:exit(0);
      break;
default:printf("invalid choice\n");
       break;
}
}
}
```

```
void push()
{
if(top==MAX-1)
printf("stack overflow");
else
{
printf("enter the item to be pushed\n");
scanf("%d",&item);
top=top+1;
stack[top]=item;
}
}
void pop()
{
if(top==-1)
printf("stack underflow");
else
{
item=stack[top];
top=top-1;
printf("deleted item is %d",item);
}
}
void display()
{
int i;
if(top==-1)
printf("stack is empty");
else
{
for(i=top;i>=0;i--)
printf("%d\t",stack[i]);
}
}
```

```
void palindrome()
{
int num[10],i=0,k,flag=1;
k=top;
while(k!=-1)
num[i++]=stack[k--];
for(i=0;i<=top;i++)
{
if(num[i]==stack[i])
continue;
else
flag=0;
}
if(top==-1)
printf("stack is empty");
else
{
if(flag)
printf("palindrome");
else
printf("not a palindrome");
}
}
```

Output

------- STACK OPERATIONS ------
1. Push
2. Pop
3. Palindrome
4. Display
5. Exit

Enter your choice 1
enter element to be inserted
10
------- STACK OPERATIONS------
1. Push
2. Pop
3. Palindrome
4. Display
5. Exit

Enter your choice 1
enter element to be inserted
20
------- STACK OPERATIONS------
1. Push
2. Pop
3. Palindrome
4. Display
5. Exit

Enter your choice 1
enter element to be inserted
30
------- STACK OPERATIONS------
1. Push
2. Pop
3. Palindrome
4. Display
5. Exit
Enter your choice 1
enter element to be inserted
40
------- STACK OPERATIONS------
1. Push
2. Pop
3. Palindrome
4. Display
5. Exit
Enter your choice 1
enter element to be inserted
50
------- STACK OPERATIONS------
1. Push
2. Pop
3. Palindrome
4. Display
5. Exit
Enter your choice 1
Stack Overflow:
------- STACK OPERATIONS------
1. Push
2. Pop
3. Palindrome
4. Display
5. Exit
Enter your choice 4

stack elements are:
50 40 30 20 10

------- STACK OPERATIONS------
1.Push
2.Pop
3. Palindrome
4. Display
5. Exit
Enter your choice 2
The poped element: 50
------- STACK OPERATIONS------
1.Push
2.Pop
3. Palindrome
4. Display
5. Exit
Enter your choice 2
The poped element: 40
------- STACK OPERATIONS------
1.Push
2.Pop
3. Palindrome
4. Display
5. Exit
Enter the choice
2
The poped element: 30
------- STACK OPERATIONS------
1. Push
2. Pop
3. Palindrome
4. Display
5. Exit
Enter your choice 2
The poped element: 20

------- STACK OPERATIONS------

Push

1. Pop
2. Palindrome
3. Display
4. Exit

Enter the choice

2

The poped element: 10

------- STACK OPERATIONS------

1. Push
2. Pop
3. Palindrome
4. Display
5. Exit

The enter the choice 2

Stack is Empty

Program outcome :

- Analyze the stack overflow and underflow conditions.
- Identify different application ofstacks.
- Implement to check palindrome numbers using stacks.
- Familiarized with push and pop operations on stack.

Viva Questions:

- What is Stack and where it can be used?
- What is the difference between PUSH and POP?
- Differentiate STACK from ARRAY.
- What is the difference between a stack and a Queue?

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 assignelements in the stack to the postfixexpression until you 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

Operator	priority
#	0
(	1
+ -	2
* / %	3
^	4

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<ctype.h>
#define SIZE 50
char s[SIZE];
int top=-1;
 void push(char elem)
 {
  s[++top]=elem;
 }
char pop()
{
return s[top--];
}
int pr(char elem)
{
switch(elem)
{
case '#':return 0;
case '(':return 1;
case '+':
case '-':return 2;
case '*':
case '/':
case '%':return 3;
case '^':return 4;
}
}
void main()
{
charinfix[50],postfix[50],ch,elem;
int i=0,k=0;
printf("enter the  infix expression\n");
gets(infix);
push('#');
while((ch=infix[i++])!='\0')
```

```
{
if(ch=='(')
push(ch);
else if(isalnum(ch))
postfix[k++]=ch;
else if(ch==')')
{
while(s[top]!='(')
postfix[k++]=pop();
elem=pop();
}
else
{
while(pr(s[top])>=pr(ch))
postfix[k++]=pop();
push(ch);
}
}
while(s[top]!='#')
postfix[k++]=pop();
postfix[k]='\0';
printf("infix expression is %s\n postfix expression is %s\n",infix,postfix);
}
```

Output1

enter the Infix Expression
((a+b)*c)
Given Infix Expn is: ((a+b)*c)
The Postfix Expn is: ab+c*

Output 2

enter the Infix Expression
(a+ (b-c)*d)
Given Infix Expn is: (a+ (b-c)*d)
The Postfix Expn is: abc-d*+

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 inC for the following Stack Applications

a. **Evaluation of Suffixexpression withsingle digit operands and 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 precedence and associativity.

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<stdlib.h>
#include<string.h>
#include<math.h>
#define MAX 50
char post[MAX];
int stack[MAX],top=-1,i;
void pushstack(int);
void calculator(char);
main()
{
 printf("enter suffix expression\n");
 gets(post);
 for(i=0; i<strlen(post); i++)
 {
  if(post[i]>'0'&& post[i]<='9')
  pushstack(i);
  else
   calculator(post[i]);
  }
   printf("result=%d\n",stack[top]);
 }
void pushstack(int i)
 {
   top=top+1;
   stack[top]=(int)(post[i]-48);
 }
void calculator(char c)
 {
  int a,b,ans;
  b=stack[top--];
  a=stack[top--];
```

```
 switch(c)
 {
  case '+':ans=a+b;break;
  case '-':ans=a-b;break;
  case '*':ans=a*b;break;
  case '/':ans=a/b;break;
  case '%':ans=a%b;break;
  case '^':ans=pow(a,b);break;
  default :printf("wrong input\n");
  exit(0);
 }
  top++;
  stack[top]=ans;
}
```

Output1

enter suffix expression:
23+
The result is 5

Output2

enter suffix expression:
123-4*+
The result is -3.

Output3

enter suffix expression:

623+-382/+*2$3+

The result is 52

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 as parameter,

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 may be placed on top of a smaller disk.

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

PROGRAM:

```
#include<stdio.h>
Void tower(int n,char frompeg,char topeg,char auxpeg); int
n;
void main()
{
 printf("Enter the no. of discs: \n");
 scanf("%d",&n);
 printf("the number of moves in tower of henoi problem\n");
 tower(n,'A','C','B');
}
void tower(int n,char frompeg,char topeg,char auxpeg)
{
 if(n==1)
  {
   printf("move disk1 from %C to %C\n ",frompeg,topeg);
   return;
  }
 tower(n-1,frompeg,auxpeg,topeg);
 printf("move disk%d from %C to %C\n",n,frompeg,topeg);
 tower(n-1,auxpeg,topeg,frompeg);
}
```

Output

```
Enter the no. of discs:
3
the number of moves in tower of henoi problem
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
```

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 usingrecursion?
- 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 circularqueue
- Know the advantages of circular queue over liner queue.
- Understand the insertion and deletion operation oncircular queue.
- Understand overflow and underflow conditions incircularqueue.

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.

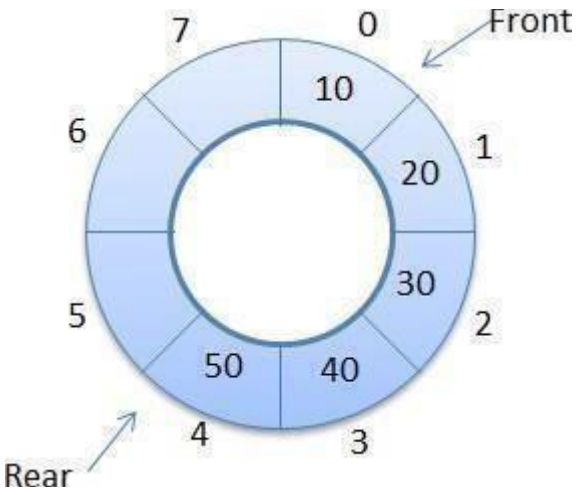

Fig6-circular queue

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
   Char  q[MAX],item;
int  f=0,r=-1,count=0;
void insert();
void  delete();
void  display();
main()
{
int ch;
while(1)
{
printf("1.insert 2.delete 3.display 4.exit \n");
printf("enter   choice\n");
scanf("%d",&ch);
switch(ch)
{
case  1:getchar();insert();
      break;
case  2:delete();
      break;
case  3:display();
      break;
case 4:exit(0);
default  :printf("Invalid  choice\n");
        break;
```

```
}
}
}
void insert()
{
if(count==MAX)
printf("queue   overflow\n");
else
{
printf("enter the item to be inserted\n");
scanf("%c",&item );
r=(r+1)%MAX;
q[r]=item;
count++;
}
}
void delete()
{
if(count==0)
printf("queue   underflow\n");
else
{
printf("deleted item is %c\n",q[f]);
f=(f+1)%MAX;
count--;
}
}
void display()
{
int j=f,i;
```

```
if(count==0)
printf("queue is empty\n");
else
{
printf("contents of circular queue\n");
for(i=1;i<=count;i++)
{
printf("%c\t",q[j]);
j=(j+1)%MAX;
}
printf("total number of items=%d\n",count);
}
}
```

Output

```
1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 1
Enter the item to be inserted: A
1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 1
Enter the item to be inserted: B
1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 1
Enter the item to be inserted: C
1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 1
Enter the item to be inserted: D
1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 3
Contents of Queue is:
A B C D
1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 1
Enter the character / item to be inserted: F
Queue is Full
1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 2
Deleted item is: A
1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 2
Deleted item is: B
1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 3
Contents of Queue is:
C D
1. Insert 2. Delete 3. Display 4. Exit
Enter the choice: 1
Enter the item to be inserted: K
1. Insert 2. Delete
Enter the choice: 3
```

Contents of Queue is:
C D K
1. Insert 2. Delete 3. Display4.Exit
Enter the 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 and Deletion at End of SLL**
d **Perform Insertion and Deletion at Front of SLL**
e **Demonstrate how this SLL can be used as STACKandQUEUE**
f **Exit**

Program objective:

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

Algorithm

Step 1: declare structure of node createemptylist
head->null
Step2: **Insert at front end**
head<-null
returntemp
if list is empty
temp->link=head
return head
Step 3**:Insert at rear end**
head=null
returntemp
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->link

free(cur)

Step 5:**Delete at Rear end**

head->link=null

return 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<stdio.h>
#include<stdlib.h>
#include<string.h>
void create();
void insert_front();
void insert_rear();
void display();
void delete_front();
void delete_rear();
int count=0;
struct node{
char usn[20],name[50],branch[10];
int sem;
unsigned long long int phno;
struct node *link;
};
struct node *first=NULL,*last=NULL,*temp=NULL,*p;
void main()
{
int ch,n,i;
while(1)
{
printf("1.create SLL 2.insert at front 3.insert at rear 4.display 5.delete at front 6.delete at
rear 7.exit\n");
printf("enter choice\n");
scanf("%d",&ch);
switch(ch)
{
```

```
case 1:printf("enter the no.of students\n");
 scanf("%d",&n);
   for(i=1;i<=n;i++)
       insert_front();
      break;
case  2:insert_front();
break;
case  3:insert_rear();break;
case 4:display();break;
case  5:delete_front();break;
case  6:delete_rear();break;
case 7:exit(0);
default:printf("invalid     choice\n");break;
}
}
}
void create()
{
char  usn[20],name[50],branch[10];
int sem;
unsigned long long int phno;
temp=(struct  node*)malloc(sizeof(struct  node));
printf("enter   usn,name,branch,sem,phno\n");
scanf("%s%s%s%d%llu",usn,name,branch,&sem,&phno);
strcpy(temp->usn,usn);
strcpy(temp->name,name);
strcpy(temp->branch,branch);
temp->sem=sem;
temp->phno=phno;
count++;
```

```
}
void insert_front()
{
if(first==NULL)
{
create();
temp->link=NULL;
first=temp;
last=temp;
}
else
{
create();
temp->link=first;
first=temp;
}
}
void insert_rear()
{
if(first==NULL)
{
create();
temp->link=NULL;
first=temp;
last=temp;
}
else
{
create();
```

```
temp->link=NULL;
last->link=temp;
last=temp;
}
}
void display()
{
if(first==NULL)
{
printf("list is empty\n");
}
else
{
p=first;
printf("content of list is\n");
while(p!=NULL)
{
printf("%s\t%s\t%s\t%d\t%llu\n",p->usn,p->name,p->branch,p->sem,p->phno);
p=p->link;
}
printf("total no.of students %d\n",count);
}
}
void delete_front()
{
p=first;
if(first==NULL)
{
printf("list is empty\n");
```

```
}
else if(p->link==NULL)
{
printf("deleted node is %s\t%s\t%s\t%d\t%llu\n",p->usn,p->name,p->branch,p->sem,p->phno);
free(p);
first=NULL;
count--;
}
else
{
first=p->link;
printf("deleted node is %s\t%s\t%s\t%d\t%llu\n",p->usn,p->name,p->branch,p->sem,p->phno);
free(p);
count--;
}
}
void delete_rear()
{
p=first;
if(first==NULL)
{
printf("list is empty\n");
}
else if(p->link==NULL)
{
printf("deleted node is %s\t%s\t%s\t%d\t%llu\n",p->usn,p->name,p->branch,p->sem,p->phno);
free(p);
first=NULL;
count--;
```

```
}
else
{
while(p->link!=last)
p=p->link;
    printf("deleted node is %s\t%s\t%s\t%d\t%llu\n",last->usn,last->name,last->branch,last->sem,last->phno);
free(last);
p->link=NULL;
last=p;
count--;
}
}
```

Output

```
_____MENU-_____

1  create a SLL 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 :

2
List empty to display


_____MENU-_____

1  create a SLL of n emp
2  - Display frombeginning
3 - Insert at end
4 - delete at end
5 - Insert at beg
6 - delete at beg
7 - exit
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Enter choice :1

Enter no of students :
2
```

Enter usn,name, branch, sem, phno of student :
4ad16cs022 harsha cs 3 9912367789

Enter usn,name, branch, sem, phno of student :
4ad16cs024 deepak cs 3 9538218822

_____MENU-_____

1 create a SLL of n emp
2 - Display frombeginning
3 - Insert at end
4 - delete at end
5 - Insert at beg
6 - delete at beg
7 - exit
– – – – – – – – – – – – – – – – – –

Enter choice :
2

Linked list elements from begining :
4ad16cs024 deepak cs 3 9538218822
4ad16cs022 harsha cs 3 9912367789
No of students = 2
_____MENU-_____

1 create a SLL of n emp
2 - Displayfrombeginning
3 - Insert at end
4 - delete at end
5 - Insert at beg
6 - delete at beg
7 - exit
– – – – – – – – – – – – – – – – – –

Enter choice :
3

Enter usn,name, branch, sem, phno of student :
4ad16cs011 bharath cs 3 9912698467

MENU-

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

– – – – – – – – – – – – – – – – – – –

Enter choice :
2

Linked list elements from begining :

4ad16cs024	deepak	cs	3	9538218822
4ad16cs022	harsha	cs	3	9912367789
4ad16cs011	bharath	cs	3	9912698467

No of students = 3

MENU-

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

– – – – – – – – – – – – – – – – – – –

Enter choice :
5

Enter usn,name, branch, sem, phno of student :
4ad16cs033 jayakumar cs 3 8903478345

MENU-

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

– – – – – – – – – – – – – – – – – – –

Enter choice :
2

Linked list elements from begining :

4ad16cs033	jayakumar	cs	3	8903478345
4ad16cs024	deepak	cs	3	9538218822
4ad16cs022	harsha	cs	3	9912367789
4ad16cs011	bharath	cs	3	9912698467

No of students = 4

MENU-

1 create a SLL of n emp
2 - Displayfrombeginning
3 - Insert at end
4 - delete at end
5 - Insert at beg
6 - delete at beg
7 - exit

– – – – – – – – – – – – – – – – – – –

Enter choice :
4
4ad16cs011 bharath cs 3 9912698467

MENU-

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

– – – – – – – – – – – – – – – – – –

Enter choice :
2

Linked list elements from begining :

4ad16cs033	jayakumar	cs	3	8903478345
4ad16cs024	deepak	cs	3	9538218822
4ad16cs022	harsha	cs	3	9912367789

No of students = 3

MENU-

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

– – – – – – – – – – – – – – – – – –

Enter choice :
6
4ad16cs033 jayakumar cs 3 8903478345

MENU-

1 create a SLL of n emp
2 - Display frombeginning
3 - Insert at end
4 - delete at end
5 - Insert at beg
6 - delete at beg
7 - exit
– – – – – – – – – – – – – – – – – – –

Enter choice :
2

Linked list elements from begining :
4ad16cs024 deepak cs 3 9538218822
4ad16cs022 harsha cs 3 9912367789
No of students = 2

MENU-

1 create a SLL of n emp
2 - Displayfrombeginning
3 - Insert at end
4 - delete at end
5 - Insert at beg
6 - delete at beg
7 - exit
– – – – – – – – – – – – – – – – – – –

```
Enter choice :
8
wrong choice

_____MENU-_____

1  create a SLL of n emp
2  - Display frombeginning
3 - Insert at end
4 - delete at end
5 - Insert at beg
6 - delete at beg
7 - exit
– – – – – – – – – – – – – – – – – – –

Enter choice :
7
```

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 methodology to 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=null
returntemp
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=head->rlink

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=cur;
cur=cur->rlink;
}
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<stdio.h>
#include<stdlib.h>
#include<string.h>
void create(); void
insert_front();  void
insert_rear(); void
display();
void  delete_front();
void  delete_rear();
int  count=0;
struct  node
{
     int ssn;
     char   name[50],dept[20],desg[20];
     float sal;
     unsigned long long int phno;
     struct node *llink,*rlink;
};
struct  node  *first=NULL,*last=NULL,*temp;
main()
{
int ch,n,i;
while(1)
{
printf("1.create\n 2.insert_front\n 3.insert_rear\n 4.display\n 5.delete_front\n 6.delete_rear\n
7.exit\n");
printf("enter    choice\n");
```

```
scanf("%d",&ch);
switch(ch)
{
case 1:printf("enter the number of employee\n");
      scanf("%d",&n);
      for(i=0;i<n;i++)
      insert_rear();
      break;
case   2:insert_front();break;
case   3:insert_rear();break;
case 4:display();break;
case   5:delete_front();break;
case   6:delete_rear();break;
case  7:exit(0);
default:printf("invalid      choice\n");break;
}
}
}
void create()
{
      int ssn;
      char   name[50],dept[20],desg[20];
      float sal;
 unsigned long long int phno;
      temp=(struct  node*)malloc(sizeof(struct   node));
      temp->llink=temp->rlink=NULL;
printf("enter   ssn,name,dept,desg,salaryand   phno\n");
      scanf("%d%s%s%s%f%llu",&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 insert_front()
{
     if(first==NULL)
     {
              create();
              first=temp;
              last=temp;
     }
     else
     {
              create();
    temp->rlink=first;
              first->llink=temp;
              first=temp;
     }
}
void insert_rear()
     {
      if(first==NULL)
              {
              create();
              first=temp;
```

```
            last=temp;
            }
    else
    {
            create();
            last->rlink=temp;
            temp->llink=last;
            temp->rlink=NULL;
            last=temp;
    }
}
void display()
{
struct node *p;
if(first==NULL)
{
printf("list is empty\n");
return;
}
p=first;
printf("contents of list\n");
while(p!=NULL)
{
printf("%d\t%s\t%s\t%s\t%f\t%llu\n",p->ssn,p->name,p->dept,p->desg,p->sal,p->phno);
p=p->rlink;
}
printf("total no. of employee %d\n",count);
}
```

```
void delete_front()
{
struct node *p;
if(first==NULL)
{
printf("list is empty,cannot delete\n");
}
else  if(first->rlink==NULL)
{
printf("deleted data is %d\t%s\t%s\t%s\t%f\t%llu\n",first->ssn,first->name,first->dept,first->desg,first->sal,first->phno);
first=NULL;
free(first);
count--;
}
else
{
p=first;
first=p->rlink;
printf("deleted data is %d\t%s\t%s\t%s\t%f\t%llu\n",p->ssn,p->name,p->dept,p->desg,p->sal,p->phno);
free(p);
count--;
}
}
void delete_rear()
{
  struct node*p;
  if(first==NULL)
            {
```

```
            printf("list is empty,cannot delete\n");
            }
    else   if(first->rlink==NULL)
            {
            printf("deleted data is %d\t%s\t%s\t%s\t%f\t%llu\n",first->ssn,first->name,first->dept,first->desg,first->sal,first->phno);
            first=NULL;
            free(first);
            count--;
            }
    else
    {
    p=last;
    last=p->llink;
printf("deleted data is %d\t%s\t%s\t%s\t%f\t%llu\n",p->ssn,p->name,p->dept,p->desg,p->sal,p->phno);
    free(p);
    last->rlink=NULL;
    count--;
    }
}
```

Output

```
_____MENU-_____

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

Enter choice :
2
List empty to display

_____MENU-_____

1- create a DLL of n emp
2 - Display frombeginning
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 ofemployee :
120 harsha cs instructor 14000 9912378956
```

Enter ssn,name,department, designation, salary and phno ofemployee :
121 sanjay cs programmer 15000 9538215567

MENU-

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

Enter choice :
2

Linked list elements from begining :
120 harsha cs instructor 14000.000000 9912378956
121 sanjay cs programmer 15000.000000 9538215567
No of employees = 2

MENU-

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

Enter choice :
3

Enter ssn,name,department, designation, salary and phno ofemployee :
123 deepak cs instructor 14000 9534567812

_____MENU-_____

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

Enter choice :
2

Linked list elements from begining :
120 harsha cs instructor 14000.000000 9912378956
121 sanjay cs programmer 15000.000000 9538215567
123 deepak cs instructor 14000.000000 9534567812
No of employees = 3
_____MENU-_____

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

Enter choice :
5

Enter ssn,name,department, designation, salary and phno ofemployee :
124 lohith cs lecturer 20000 9967834578

MENU-

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

Enter choice :
2

Linked list elements from begining :
124 lohith cs lecturer 20000.000000 9967834578
120 harsha cs instructor 14000.000000 9912378956
121 sanjay cs programmer 15000.000000 9538215567
123 deepak cs instructor 14000.000000 9534567812
No of employees = 4

MENU-

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

Enter choice :
4
123 deepak cs instructor 14000.000000 9534567812

MENU-

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

Enter choice :
2

Linked list elements from begining :
124 lohith cs lecturer 20000.000000 9967834578
120 harsha cs instructor 14000.000000 9912378956
121 sanjay cs programmer 15000.000000 9538215567
No of employees = 3

MENU-

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

Enter choice :
6

124lohith cs lecturer 20000.000000 9967834578

MENU-

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

-

Enter choice :
2

Linked list elements from begining :
120 harsha cs instructor 14000.000000 9912378956
121 sanjay cs programmer 15000.000000 9538215567
No of employees = 2

MENU-

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

-

Enter choice :
8
wrong choice
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 :
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 usingSCLL.
- Understand the methodology to add twopolynomialusingSCLL.

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.

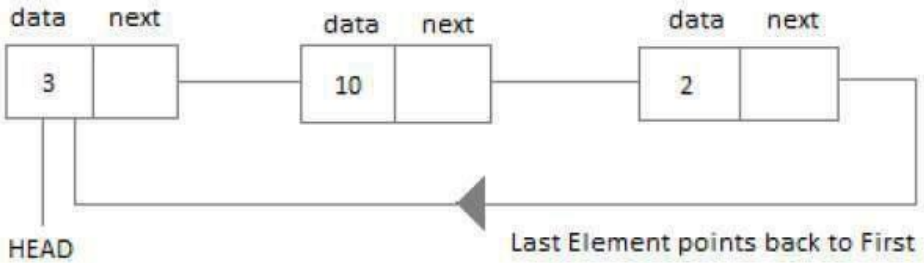

Fig-8- Circular Linked List

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
struct node
{
  int co,ex,ey,ez;
  struct node *link;
};
typedef struct node NODE;
NODE *createnode(int,int,int,int);
NODE    *attachnode(NODE*,NODE*);
NODE *readpoly();
void display(NODE*);
void evaluate(NODE*);
NODE    *addpoly(NODE*,NODE*,NODE*);

NODE *createnode(int co,int ex,int ey,int ez)
{
  NODE *temp;

temp=(NODE*)malloc(sizeof(NODE));
  temp->co=co;
  temp->ex=ex;
temp->ey=ey;
temp->ez=ez;
temp->link=NULL;
return temp;
}
```

```
NODE *attachnode(NODE *temp,NODE *head)
{
  NODE *cur;
  cur=head->link;
  while(cur->link!=head)
{
  cur=cur->link;
}
cur->link=temp;
temp->link=head;
return head;
}
NODE *readpoly()
{
  int i,n,co,ex,ey,ez;
  NODE    *head=(NODE*)malloc(sizeof(NODE));
  NODE *temp;

  head->link=head;
  printf("enter the number of terms\n");

scanf("%d",&n);
  for(i=0;i<n;i++)
  {
    printf("term %d\n",i+1);
    printf("enter the coefficient\n");
    scanf("%d",&co);
    printf("enter exponent values of x,yand z\n");
    scanf("%d%d%d",&ex,&ey,&ez);
```

```
temp=createnode(co,ex,ey,ez);

head=attachnode(temp,head);
  }
  return head;
}

void display(NODE *poly)
{
  NODE *cur;
cur=poly->link;

while(cur!=poly)
  {
    printf("%dx^%dy^%dz^%d+",cur->co,cur->ex,cur->ey,cur->ez);
    cur=cur->link;
  }
  printf("\n");
}

void evaluate(NODE *poly)
{
  NODE *cur;
  int x,y,z,res=0;
  cur=poly->link;
  printf("enter the values of x,y,z\n");
  scanf("%d%d%d",&x,&y,&z);
while(cur!=poly)
  {
```

```
    res+=cur->co*pow(x,cur->ex)*pow(y,cur->ey)*pow(z,cur->ez);
    cur=cur->link;
  }

printf("result=%d\n",res);
}

NODE *addpoly(NODE *p1,NODE *p2,NODE *poly)
{
  int comp;
  NODE *a,*b,*temp;
  a=p1->link;
  b=p2->link;
  while(a!=p1&&b!=p2)
  {
    if(a->ex==b->ex && a->ey==b->ey && a->ez==b->ez)
    comp=0;
    else if(a->ex>b->ex)
    comp=1;
    else if(a->ex==b->ex && a->ey==b->ey)
    comp=1;
    else if(a->ex==b->ex && a->ey==b->ey && a->ez>b->ez)
    comp=1;
    else
    comp=-1;
    switch(comp)
    {
      case 0:temp=createnode(a->co+b->co, a->ex, a->ey, a->ez);
             poly=attachnode(temp,poly);
             a=a->link;
```

```
            b=b->link;
            break;
      case 1:temp=createnode(a->co, a->ex,a->ey,a->ez);

poly=attachnode(temp,poly);
            a=a->link;
            break;
     case-1:temp=createnode(b->co,b->ex,b->ey,b->ez);
poly=attachnode(temp,poly);
            b=b->link;
            break;
    }
  }
  while(a!=p1)
  {
    temp=createnode(a->co,a->ex,a->ey,a->ez);
poly=attachnode(temp,poly);
    a=a->link;
  }
  while(b!=p2)
  {    temp=createnode(b->co,b->ex,b->ey,b->ez);

poly=attachnode(temp,poly);
    b=b->link;
  }
  return poly;
}
main()
{
  int ch;
```

```
 NODE *p1,*p2,*p3;
 p3=(NODE*)malloc(sizeof(NODE));
 p3->link=p3;
 while(1)
 {
  printf("1.represent and evaluate 2.add two polynomial 3.exit\n");
  printf("enter choice\n");
  scanf("%d",&ch);
  switch(ch)
  {
    case 1:printf("enter a polynomial\n");
p1=readpoly();
    display(p1);
      evaluate(p1);
          break;
    case 2:printf("enter polynomial 1\n");
p1=readpoly();
          display(p1);
          printf(" enter polynomial 2\n");
          p2=readpoly();
          display(p2);
          p3=addpoly(p1,p2,p3);
          printf("the resultant polynomial is\n");
display(p3);
          break;
    case 3:exit(0);
    default:printf("invalid choice\n");
           break;
}
 } }
```

Output

<< MENU >>
Polynomial Operations : 1.Add 2.Evaluate
3.Exit
Enter your choice==>1
Enter no of terms of polynomial==>3
Enter coef& expo==>
4
3
Enter coef& expo==> 22
Enter coef& expo==> 51
The polynomial is==>5x^(1) + 2x^(2) + 4x^(3)
Enter no of terms of polynomial==>3
Enter coef& expo==> 4 1
Enter coef& expo==>
3
2
Enter coef& expo==> 5 3
The polynomial is==>4x^(1) + 3x^(2) + 5x^(3)
Addition of polynomial==>
The polynomial is==>9x^(1) + 5x^(2) + 9x^(3)
Enter your choice==>2
Enter no of terms of polynomial==>3
Enter coef& expo==>
3
1

Enter coef& expo==> 42
Enter coef& expo==> 54
The polynomial is==>3x^(1) + 4x^(2) + 5x^(4)
Enter the value of x=2
Value of polynomial=102
Enter your choice==>3
exit

Program outcome :

- Implement Singly Circular Linked List (SCLL) using header node.
- Identify the application of SCLL.
- 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 In-order, Preorder and PostOrder

C Searchthe BST for a givenelement (KEY) and reportthe appropriate message

D Delete anelement (ELEM) from BST

E 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 ofa node has key less than or equalto its parent node's V key.
- The right sub-tree of a node has key greater than or equal to its parent node'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)

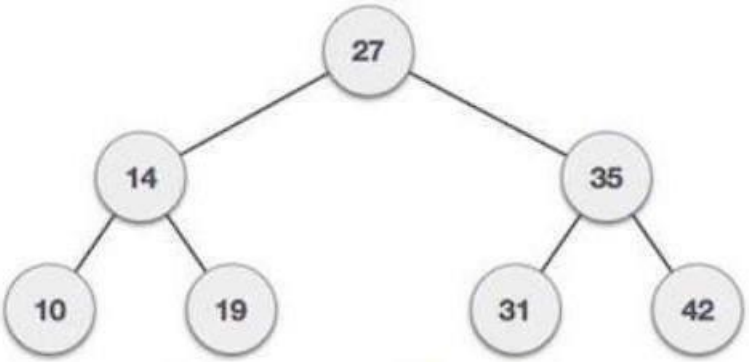

Fig: An example of BST

Fig 9-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 inorder manner.
- **Postorder Traversal** − traverse a tree in a postorder manner.

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
typedef struct bst
{
int data;
struct bst *rchild,*lchild;
}node;
node*  getnode();
void insert(node*,node*);
void inorder(node*);
void preorder(node*);
void postorder(node*);
int search(node*,int);
int n;
node*  getnode()
{
node *temp;
temp=(node*)malloc(sizeof(node));
temp->lchild=NULL;
temp->rchild=NULL;
return temp;
}

void main()
{
int ch;
int key,ans=1;
```

```
node*newnode,*root,*temp,*parent;
root=NULL;
while(1)
{
printf("1.create 2.search 3.travers 4.exit\n");
printf("enter choice\n");
scanf("%d",&ch);
switch(ch)
{
case 1:while(ans==1)
       {

newnode=getnode();
       printf("enter the  element\n");
       scanf("%d",&newnode->data);
       if(root==NULL)
       root=newnode;
       else

insert(root,newnode);
       printf("want to continue[0/1]\n");
       scanf("%d",&ans);
       }
       break;
case 2:printf("enter the element to search\n");
       scanf("%d",&key);
       search(root,key);
       if(n==1)
       printf("search successful\n");
       else
```

```
        printf("search unsuccessful\n");
       break;

case 3:if(root==NULL)
        printf("tree is empty\n");
        else
        {
       printf("\n inorder traversal\n");
         inorder(root);
     printf("\n postorder traversal\n");
         postorder(root);
       printf("\n preorder traversal\n");
         preorder(root);

         }
         break;
case 4:exit(0);break;
default :printf("wrong choice\n");
       break;
}
}
}
void insert(node *root,node *newnode)
{
if(root->data<newnode->data)
{
if(root->rchild==NULL)
root->rchild==newnode;
else
insert(root->rchild,newnode);
```

```
}
else
{
if(root->lchild==NULL)
root->lchild=newnode;
else
insert(root->lchild,newnode);
}
}
void inorder(node *ptr)
{
if(ptr!=NULL)
{
inorder(ptr->lchild);
printf("%d\t",ptr->data);
inorder(ptr->rchild);
}
}
void preorder(node*ptr)
{
if(ptr!=NULL)
{
printf("%d\t",ptr->data);
preorder(ptr->lchild);
preorder(ptr->rchild);
}
}
void postorder(node *ptr)
{
if(ptr!=NULL)
```

```
{
postorder(ptr->lchild);
postorder(ptr->rchild);
printf("%d\t",ptr->data);
}
}
int search(node*root,int key)
{
node *temp;
temp=root;
if(root!=NULL)
{
if(temp->data==key)
n=1;
else if (key>temp->data)
search(temp->rchild,key);
else
search(temp->lchild,key);
}
else n=0;
return n;
}
```

Output

program for binary search tree
1.Create
2. Search
3. RecursiveTraversals
4.Exit
Enter your Choice= 1
Enter the element=15
Want to entermoreelements?(1/0)1
Enter the element=25
Want to entermoreelements?(1/0)1
Enter the element=35
Want to entermoreelements?(1/0)1
Enter the element=45
Want to entermoreelements?(1/0)1
Enter the element=5
Want to entermoreelements?(1/0)1
Enter the element=7
Want to enter more elements?(1/0)0
1.Create
2. Search
3. RecursiveTraversals
4.Exit
Enter your choice=2
Enter elements to be searched=7
The 7 element is present
parent of node 7 is 5
1.Create
2. Search
3. RecursiveTraversals
4.Exit
Enter your choice=2
Enter elements to be searched=88
the 88 element is not present

1.Create
2.Search
3.Recursive Traversals
4.Exit
Enter your choice=3
The inorder display =5 7 15 25 35 45
The preorder display=15 5 7 25 35 45
The postorder display=7 5 45 35 25 15
1.Create
2. Search
3. RecursiveTraversals
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

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 ina digraph using BFS method**

c **Check whether a given graph is connected or notusingDFS 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 proceeds to 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    n,a[10][10],i,j,source,s[10],choice,count;
void bfs(int n,int a[10][10],int source,int s[])
{
    int q[10],u;
    int front=1,rear=1;
    s[source]=1;
    q[rear]=source;
    while(front<=rear)
    {
        u=q[front];
        front=front+1;
        for(i=1;i<=n;i++)
        if(a[u][i]==1&&s[i]==0)
        {
            rear=rear+1;
            q[rear]=i;
            s[i]=1;
        }
    }
}
void dfs(int n,int a[10][10],int source,int s[])
{
   s[source]=1;
   for(i=1;i<=n;i++)
     if(a[source][i]==1 && s[i]==0)
     dfs(n,a,i,s);
}
int main()
{
    printf("Enter the number of nodes : \n");
    scanf("%d",&n);
    printf("\n Enter the adjacency matrix\n");
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
```

```
        scanf("%d",&a[i][j]);
while(1)
  {

   printf("\n\n1.BFS\n 2.DFS\n 3.Exit\n");
   printf("\nenter your choice\n");
   scanf("%d",&choice);
   switch(choice)
   {
    case 1: printf("\n Enter the source :\n");
          scanf("%d",&source);
          for(i=1;i<=n;i++)
          s[i]=0;
            bfs(n,a,source,s);
            for(i=1;i<=n;i++)
            {
            if(s[i]==0)
              printf("\n The node %d is not reachable\n",i);
            else
              printf("\n The node %d is reachable\n",i);
            }
            break;
   case 2:printf("\nEnter the source vertex :\n");
        scanf("%d",&source);
     count=0;
    for(i=1;i<=n;i++)
    s[i]=0;
     dfs(n,a,source,s);
     for(i=1;i<=n;i++)
     if(s[i])
          count=count+1;
     if(count==n)
          printf("\nThe graph is connected.");
     else
          printf("\nThe graph is not connected.");
     break;
   case 3: exit(0);
```

```
    }
}
}
```

Output1

```
Enter the number of nodes
: 4

 Enter the  adjacencymatrix
0 0 1  0
1 0 1 0
0 0 0 0
0 0 0 0


1. BFS
 2. DFS
 3. Exit

enter your choice
1

 Enter the source :
1

 The node 1 is reachable

 The node 2 is notreachable

 The node 3 is reachable

 The node 4 is notreachable

1. BFS
 2. DFS
 3. Exit

enter your choice
2
```

Enter the source vertex :
1

The graph is not connected.

1.BFS
2. DFS
3. Exit
enter your choice
3

Output2

Enter the number of nodes :
3

Enter the adjacencymatrix
0 1 1
0 0 0
0 0 0

1. BFS
2. DFS
3. Exit

enter your choice
1

Enter the source :
1

The node 1 is reachable

The node 2 is reachable

The node 3 is reachable

1. BFS
2. DFS
3. Exit

enter your choice
2

Enter the source vertex:
1

The graph is not connected.

1.BFS
2. DFS
3. Exit

enter your choice
3

Output3

Enter the number of nodes
: 3

Enter the adjacencymatrix
0 1 0
0 0 1
1 0 0

1. BFS
2. DFS
3. Exit

enter your choice
1

Enter the source :
1

The node 1 is reachable

The node 2 is reachable

The node 3 is reachable

1. BFS
2. DFS
3. Exit

enter your choice
2

Enter the source vertex:
1

The graph is connected.

1.BFS
2. DFS
3. Exit

enter your choice
3

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 mmemory 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
void display(int a[MAX]);
int create(int num);
void linearprob(int a [MAX],int key,int num);
void main()
{
int   a[MAX],i,num,key,ans=1;
printf("collission handling by linear probing\n");
for(i=0;i<MAX;i++)
a[i]= -1;
do
{
printf("enter the data\n");
scanf("%4d",&num);
key=create(num);
linearprob(a,key,num);
printf("do yuou want to continue[1/0]\n");
scanf("%d",&ans);
}while(ans);
display(a);
}
int create(int num)
{
int key;
key=num%100;
return key;
}
```

```
void linearprob(int a[MAX],int key, int num)
{
int flag=0,count=0,i;
if(a[key]==-1)
a[key]=num;
else
{
printf("\n collision deleted\n");
i=0;
while((i<key)&&(flag==0))
{
if(a[i]==-1)
{
a[i]=num;
flag=1;
break;
}
i++;
}
}
}
void display(int a[MAX])
{
int ch,i;
printf("\n 1.display all 2.filtered display\n");
printf("enter choice\n");
scanf("%d",&ch);
if(ch==1)
{
for(i=0;i<MAX;i++)
printf("%d\t%d\n",i,a[i]);
}
```

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

Output

collision handling by linear probing :
Enter the data1234
Do you wish to continue ? (1/0)1
Enter the data2548
Do you wish to continue ? (1/0)1
Enter the data3256
Do you wish to continue ? (1/0)1
Enter the data1299
Do you wish to continue ? (1/0)1
Enter the data1298
Do you wish to continue ? (1/0)1
Enter the data1398
Collision Detected...!!!
Collision avoided successfully using LINEAR PROBING
Do you wish to continue ? (1/0) 0
1.Display ALL
2.Filtered Display
the hash table is
0 1398
341234
482548
563256
981298
991299

Program outcome:

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

Viva Questions:

- What is Hashing?
- What is Linear Probing?