Bapuji Educational Association(Regd.)

Bapuji Institute of Engineering and Technology, Davangere-577004

(AN AUTONOMOUS INSTITUTE AFFILIATED TO VTU BELAGAVI, APPROVED BY AICTE, NEW DELHI)

ACCREDITED BY NBA & NAAC WITH 'A' GRADE

Department of Computer Science and Engineering

Lab Manual

3rd Sem

Data Structures Lab(BCSPCL306)

Prepared by,
Dr. Gururaj T
Associate Professor,
CSE, BIET, Davangere-04

1. Develop a Program in C to 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). Support the program with 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.

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

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

// Function to create each day in the calendar
void create(struct Day *calendar[], int n)
{        int i;
    for (i=0; i<n; i++)
    {
      // Dynamically allocate memory for each day
      calendar[i] = (struct Day*) malloc(sizeof(struct Day));
    }
}


// Function to read data into the calendar
void read(struct Day *calendar[], int n)
{ int i;
    for ( i = 0; i < n; i++) {
      //char dayName[20];
      //char activityDescription[100];
       calendar[i]->name=(char*) malloc(20);
       calendar[i]->activity=(char*) malloc(20);
      printf("Enter the name of day %d: ", i + 1);
      scanf("%s", calendar[i]->name);

      printf("Enter the date");
      scanf("%d", &calendar[i]->date);

      printf("Enter the activity");
      scanf(" %s", calendar[i]->activity);  // To read string with
spaces

      // Dynamically allocate memory for the name and activity strings
      //calendar[i]->name = (char*) malloc(20);
       //   calendar[i]->activity = (char*) malloc(20);
      // Copy the input strings to the allocated memory
       //   strcpy(calendar[i]->name, dayName);
       //   strcpy(calendar[i]->activity, activityDescription);
    }
}

// Function to display the calendar
void display(struct Day *calendar[], int n)
{
    int i;
    printf("\nWeek's Activity Details:\n");
      for ( i = 0; i < n; i++)
      {
      printf("Day: %s, Date: %d, Activity: %s\n", calendar[i]->name,
calendar[i]->date, calendar[i]->activity);
    }
}

// Main function
int main()
{

    int n = 2,i;  // Number of days in a week
    struct Day *calendar[2];
    clrscr();

    // Create the calendar
    create(calendar, n);

    // Read data into the calendar
    read(calendar, n);

    // Display the calendar
    display(calendar, n);

    // Free dynamically allocated memory
    for ( i = 0; i < n; i++)
    {
      free(calendar[i]->name);
      free(calendar[i]->activity);
      free(calendar[i]);
    }
   getch();
    return 0;
}
```

OUTPUT:

2. Develop a C program to perform the following string operations: Read a main string (STR), a pattern string (PAT), and a replacement string (REP). Then, perform pattern matching to find and replace all occurrences of PAT in STR with REP, if PAT exists in STR, without using built-in functions. If PAT is not found, display an appropriate message. Support the program with functions for each of the above operations.

```c
#include <stdio.h>

void readString(char s[]) {
    int i = 0;
    char c;
    while ((c = getchar()) != '\n' && c != EOF) {
        s[i++] = c;
    }
    s[i] = '\0';
}

int stringLength(char s[]) {
    int i = 0;
    while (s[i] != '\0') i++;
    return i;
}

int matchAt(char str[], char pat[], int pos) {
    int i = 0;
    while (pat[i] != '\0') {
        if (str[pos + i] != pat[i]) return 0;
        i++;
    }
    return 1;
}

void replaceAll(char str[], char pat[], char rep[], char result[]) {
    int i = 0, j = 0;
    int lenS = stringLength(str);
    int lenP = stringLength(pat);
    int lenR = stringLength(rep);
    int found = 0;

    while (i < lenS) {
        if (matchAt(str, pat, i)) {
            found = 1;
            for (int k = 0; k < lenR; k++)
                result[j++] = rep[k];
            i += lenP;  // skip the pattern
        } else {
            result[j++] = str[i++];
        }
    }
    result[j] = '\0';
    if (!found)
        printf("\nPattern NOT found. No replacement done.\n");
}

int main() {
    char STR[200], PAT[50], REP[50], RESULT[300];

    printf("Enter Main String (STR): ");
    readString(STR);

    printf("Enter Pattern String (PAT): ");
    readString(PAT);

    printf("Enter Replace String (REP): ");
    readString(REP);

    replaceAll(STR, PAT, REP, RESULT);

    printf("\nFinal Result String: %s\n", RESULT);

    return 0;
}
```

OUTPUT:

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 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.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 5
int s[MAX];
int top = -1;
void push();
void pop();
void palindrome();
void display();
void main()
{
    int ch;
    while (1)
    {

printf("\nMENU:\n1.push\n2.pop\n3.palindrome\n4.dispaly\n5.Exit\nenter
your choice");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            push();
            break;
        case 2:
            pop();
            break;
        case 3:
            palindrome();
            break;
        case 4:
            display();
            break;
        case 5:
            exit(0);
            break;

        default:
            printf("enter the valid choice\n");
        }
    }
}
```

```
void push()
{   int ele;
    if (top == MAX - 1)
    {
        printf("stack overflow\n");
        return;
    }
    printf("enter an element to be pushed\n");
    scanf("%d", &ele);
    top = top + 1;
    s[top] = ele;
    return;
}
void pop()
{
    if (top == -1)
    {
        printf("Stack Underflow\n");
        return;
    }
    printf("Element popped is %d \n", s[top]);
    top = top - 1;
    return;
}
void display()
{
    int i;
    if (top == -1)
    {
        printf("Stack is Empty\n");
        return;
    }
    printf("Stack elements are\n");
    for (i = top; i >= 0; i--)
        printf("| %d |\n", s[i]);
    return;
}
void palindrome()
{
    int flag = 1, i;
    for (i = 0; i <= top / 2; i++)
    {
        if (s[i] != s[top - i])
        {
            flag = 0;
            break;
        }
    }
    if (flag == 1)
        printf("It is palindrome\n");
    else
        printf("Not a palindrome\n");
}
```
OUTPUT:

4.Develop a Program in C for the following Stack Applications:
a. Convert an Infix Expression to a Postfix Expression. The program should support both parenthesized and free parenthesized expressions with the operators: +, -, *, /, % (Remainder), ^ (Power), and alphanumeric operands.
b. Evaluation of Suffix expression with single digit operands and operators: +, -, *, /, %, ^

4.a.
```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#define MAX_SIZE 100

char s[MAX_SIZE],token,infix[MAX_SIZE],postfix[MAX_SIZE];

int top=-1,i,j;


int push( char item)
{
    s[top + 1] = item;
    return top + 1;  // Return the updated top index
}


char pop()
{
        return s[top];
}


int isOperator(char ch) {
     return (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch ==
'%' || ch == '^');
}


int getPrecedence(char operator) {
    if (operator == '^')
        return 3;
    else if (operator == '*' || operator == '/' || operator == '%')
        return 2;
    else if (operator == '+' || operator == '-')
        return 1;
    else
        return 0;
}
```

```
void infixToPostfix(char infix[], char postfix[])
{


    for (i = 0, j = 0; infix[i] != '\0'; i++)
{
        token = infix[i];

        if (isalnum(token)) {
            postfix[j++] = token;
        }


        else if (token == '(')
        {
            top = push(token);
        }

        else if (token == ')')
         {
            while (top != -1 && s[top] != '(')
            {
                postfix[j++] = pop();
                top--;
            }

            if (top != -1 && s[top] != '(') {
                printf("Invalid Expression\n");
                exit(0);
            }

 else {
                pop();
                top--;
            }
        }

else if (isOperator(token))
{
                       while  (top  !=  -1  &&  getPrecedence(s[top])  >=
getPrecedence(token))
           {
                postfix[j++] = pop();
                top--;
            }
            top = push(token);
        }
    }

    while (top != -1) {
        postfix[j++] = pop();
        top--;
    }
    postfix[j] = '\0';
}

int main()
{

    printf("Enter infix expression: ");
    scanf("%s",infix);


    infixToPostfix(infix, postfix);

    printf("Postfix expression: %s\n", postfix);

    return 0;
}
```

OUTPUT:

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

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <ctype.h>
int i, top = -1;
int op1, op2, res, s[20];
char postfix[50], symbol;
void push(int item)
{
    top = top + 1;
    s[top] = item;
}
int pop()
{
    int item;
    item = s[top];
    top = top - 1;
    return item;
}
void main()
{
    printf("\nEnter a valid postfix expression:\n");
    scanf("%s", postfix);
    for (i = 0; postfix[i] != '\0'; i++)
    {
        symbol = postfix[i];
        if (isdigit(symbol))
        {
            push(symbol - '0');
        }
        else
        {
            op2 = pop();
            op1 = pop();
            switch (symbol)
            {
            case '+':
                push(op1 + op2);
                break;
            case '-':
                push(op1 - op2);
                break;
            case '*':
                push(op1 * op2);
                break;
            case '/':
                push(op1 / op2);
                break;
            case '%':
                push(op1 % op2);
                break;
            case '$':
            case '^':
                push(pow(op1, op2));
                break;
            default:printf("invalid operator\n");
                exit(0);
            }
        }
    }
    res = pop();
    printf("\n Result = %d", res);
}
```

OUTPUT:

5.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 onto Circular QUEUE b. Delete an Element from Circular QUEUE c. Demonstrate Overflow and Underflow situations on Circular QUEUE d. Display the status of Circular QUEUE e. Exit Support the program with appropriate functions for each of the above operations.

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <stdlib.h>
#define MAX 5
int front = 0, rear = -1, count = 0;
char cq[MAX], ele;
void insert();
void display();
void delete();
void main()
{
    int choice;
    while (1)
    {
        printf("\n\t1=>insert an element on to CIRCULAR
QUEUE\n\t2=>delete an element from CIRCULAR QUEUE\n\t3=>display the
status of CIRCULAR QUEUE\n\t4=>exit\n");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            insert();
            break;
        case 2:
            delete ();
            break;
        case 3:
            display();
            break;
        case 4:
            exit(0);
        default:
            printf("Enter a valid choice\n");
        }
    }
}
void insert()
{
    if (count == MAX)
    {
        printf("Circular Queue is full, elements cannot be
inserted\n");
        return;
    }
    rear = (rear + 1) % MAX;
    printf("\nenter the element to be inserted into the Circular
Queue\n");
   // ele = getche();
   scanf(" %c",&ele);
    cq[rear] = ele;
    count++;
}
void delete()
{
    if (count == 0)
    {
        printf("Circular Queue is empty, no elements to delete\n");
        return;
    }
    ele = cq[front];
    front = (front + 1) % MAX;
    printf("the element deleted is%c\n", ele);
    count--;
}
void display()
{
    int i;
    if (count == 0)
    {
        printf("CIRCULAR QUEUE is empty,no element to display\n");
        return;
    }
    printf("Circular Queue contents are\n");
    for (i = front; i != rear; i = (i + 1) % MAX)
    {
        printf("%c\t", cq[i]);
    }
    printf("%c", cq[i]);
}
```

OUTPUT:

6.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 an 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 the stack) e. Exit

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
struct node
{
    char usn[25], name[25], branch[25];
    int sem;
    long int phone;
    struct node *link;
};
typedef struct node *NODE;
NODE first;
int count = 0;
NODE getNode()
{
    NODE newnode;
    char usn[20], name[20], branch[20];
    int sem;
    long int phone;
     printf("\nEnter the usn, Name, Branch, sem, PhoneNo ofthe student:
\n");
    scanf("%s %s %s %d %ld", usn, name, branch, &sem, &phone);
    newnode = (NODE)malloc(sizeof(struct node));
    if (newnode == NULL)
    {
        printf("\nMemory is not available");
        exit(0);
    }
    strcpy(newnode->usn, usn);
    strcpy(newnode->name, name);
    strcpy(newnode->branch, branch);
    newnode->sem = sem;
    newnode->phone = phone;
    count++;
    return newnode;
}
void insertAtFront()
{
    NODE temp;
    temp = getNode();
    temp->link = first;
    first = temp;
    return;
}
void deleteAtFront()
{
    NODE temp;
    if (first == NULL)
    {
        printf("\nLinked list is empty");
        return;
    }
    if (first->link == NULL)
    {
              printf("\nThe  Student  node  with  usn:%s  is  deleted  ",
first->usn);
        count--;
        free(first);
        first = NULL;
        return;
    }
    temp = first;
    first = first->link;
    printf("\nThe Student node with usn:%s is deleted", temp->usn);
    count--;
    free(temp);
    return;
}
void insertAtEnd()
{
    NODE cur, temp;
    temp = getNode();
    temp->link = NULL;
    if (first == NULL)
    {
        first = temp;
        return;
    }
    if (first->link == NULL)
    {
        first->link = temp;
        return;
    }
    cur = first;
    while (cur->link != NULL)
    {
        cur = cur->link;
    }
    cur->link = temp;
    return;
}
void deleteAtEnd()
{
    NODE cur, prev;
    if (first == NULL)
    {
        printf("\nLinked List is empty");
        return;
    }
    if (first->link == NULL)
    {
            printf("\nThe  student  node  with  the  usn:%s  is  deleted\n",
first->usn);
        free(first);
        first = NULL;
        count--;
        return;
    }
    prev = NULL;
    cur = first;
    while (cur->link != NULL)
    {
        prev = cur;
        cur = cur->link;
    }
    printf("\nThe student node with the usn:%s is deleted", cur->usn);
    free(cur);
    prev->link = NULL;
    count--;
    return;
}
void displayStatus()
{
    NODE cur;
    int nodeNo = 1;
    if (first == NULL)
    {
        printf("\nNo Contents to display in SLL \n");
        return;
    }
    printf("\nThe contents of SLL: \n");
    cur = first;
    while (cur != NULL)
    {
        printf("\n||%d||", nodeNo);
        printf(" USN:%s|", cur->usn);
        printf(" Name:%s|", cur->name);
        printf(" Branch:%s|", cur->branch);
        printf(" Sem:%d|", cur->sem);
        printf(" Ph:%ld|", cur->phone);
        cur = cur->link;
        nodeNo++;
    }
    printf("\n No of student nodes is %d \n", count);
}
void stackDemoUsingSLL()
{
    int ch;
    while (1)
    {
        printf("\n~~~Stack Demo using SLL~~~\n");
                printf("\n1:Push  operation  \n2:  Pop  operation  \n3:
Display\n4:Exit \n");
        printf("\nEnter your choice for stack demo");
        scanf("%d", &ch);
```

```
        switch (ch)
        {
        case 1:
            insertAtFront();
            break;
        case 2:
            deleteAtFront();
            break;
        case 3:
            displayStatus();
            break;
        case 4:
            return;
        }
    }
}
void main()
{
    int ch, i, n;
    while (1)
    {
        printf("\n~~~Menu~~~");
        printf("\nEnter your choice for SLL operation \n");
        printf("\n1:Create SLL of Student Nodes");
        printf("\n2:DisplayStatus");
        printf("\n3:InsertAtEnd");
        printf("\n4:DeleteAtEnd");
           printf("\n5:Stack Demo using SLL(Insertion and Deletion at
Front)");
        printf("\n6:Exit \n");
        printf("\nEnter your choice:");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            printf("\nEnter the no of students: ");
            scanf("%d", &n);
            for (i = 1; i <= n; i++)
                insertAtFront();
            break;
        case 2:
            displayStatus();
            break;
        case 3:
            insertAtEnd();
            break;
        case 4:
            deleteAtEnd();
            break;
        case 5:
            stackDemoUsingSLL();
            break;
        case 6:
            exit(0);
```

```
        default:
            printf("\nEnter the valid choice");
        }
    }
}
```

OUTPUT:

7.Develop a menu-driven program in C for the following operations on a Doubly Linked List (DLL) of Employee Data, with the following fields: SSN, Name, Department, Designation, Salary, and Phone Number. a. Create a DLL of N Employees Data by using end insertion b. Display the status of DLL and count the number of nodes in it c. Perform Insertion and Deletion at End of DLL d. Perform Insertion and Deletion at Front of DLL e. Demonstrate how this DLL can be used as Double Ended Queue f. Exit

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
struct node
{
    char ssn[25], name[25], dept[10], designation[25];
    long int sal, phoneno;
    struct node *llink;
    struct node *rlink;
};
typedef struct node *NODE;
NODE first = NULL;
int count = 0;
NODE getNode()
{
    NODE newnode;
    newnode = (NODE)malloc(sizeof(struct node));

printf("\nEntertheEmployeessn,Name,Dept,Designation,Salary,PhoneNo\n");
     scanf("%s%s%s%s%ld%ld", newnode->ssn, newnode->name, newnode->dept,
newnode->designation, &newnode->sal, &newnode->phoneno);
    if (newnode == NULL)
    {
        printf("\nRunning out of memory");
        exit(0);
    }
    newnode->llink = NULL;
    newnode->rlink = NULL;
    count++;
    return newnode;
}
void insertAtFront()
{
    NODE temp;
    temp = getNode();
    if (first == NULL)
    {
        first = temp;
        return;
    }
    temp->rlink = first;
    first->llink = temp;
    first = temp;
    return;
}
```

```
void deleteAtFront()
{
    NODE temp;
    if (first == NULL)
    {
        printf("\nDoubly Linked List is empty");
        return;
    }
    temp = first;
    first = first->rlink;
    temp->rlink = NULL;
    first->llink = NULL;
    printf("\nThe employee node with the ssn:%sis deleted", temp->ssn);
    free(temp);
    count--;
    return;
}
void insertAtEnd()
{
    NODE cur, temp;
    temp = getNode();
    if (first == NULL)
    {
        first = temp;
        return;
    }
    cur = first;
    while (cur->rlink != NULL)
    {
        cur = cur->rlink;
    }
    cur->rlink = temp;
    temp->llink = cur;
    return;
}
void deleteAtEnd()
{
    NODE prev, cur;
    if (first == NULL)
    {
        printf("\nDoubly Linked List is empty");
        return;
    }
    if (first->rlink == NULL)
    {
            printf("\nThe  employee  node  with  the  ssn:%s  is  deleted",
first->ssn);
        free(first);
        first = NULL;
        count--;
        return;
    }
    prev = NULL;
    cur = first;
    while (cur->rlink != NULL)
    {
        prev = cur;
        cur = cur->rlink;
    }
    cur->llink = NULL;
    printf("\nThe employee node with the ssn:%s is deleted", cur->ssn);
    free(cur);
    prev->rlink = NULL;
    count--;
    return;
}
void displayStatus()
{
    NODE cur;
    int nodeno = 1;
    if (first == NULL)
    {
        printf("\nNo Contents to display in DLL");
        return;
    }
    printf("\nENode||SSN|Name|Department|Designation|Salary|Phone\n");
    cur = first;
    while (cur != NULL)
    {
                printf("\n%d||%s|%s|%s|%s|%ld|%ld",  nodeno,  cur->ssn,
cur->name, cur->dept, cur->designation, cur->sal, cur->phoneno);
        cur = cur->rlink;
        nodeno++;
    }
    printf("\nNo of employee nodes is%d", count);
}
void doubleEndedQueueDemo()
{
    int ch;
    while (1)
    {
        printf("\nDemo Double Ended Queue Operation");
         printf("\n1:Insert Queue Front\n2:Delete Queue Front\n3:Insert
Queue Rear\n 4:Delete Queue Rear\n5:Display Status\n6:Exit\n");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            insertAtFront();
            break;
        case 2:
            deleteAtFront();
            break;
        case 3:
            insertAtEnd();
            break;
        case 4:
            deleteAtEnd();
            break;
        case 5:
            displayStatus();
            break;
        case 6:
            return;
        default:
            printf("invalid choice\n");
        }
    }
}
void main()
{
    int ch, i, n;
    while (1)
    {
        printf("\n~~~Menu~~~");
        printf("\nEnter your choice for DLL operation\n");
        printf("\n1:Create DLL of employ Nodes");
        printf("\n2:Display Status");
        printf("\n3:double Ended Queue Demo");
        printf("\n4:Exit\n");
        printf("\nEnter your choice:");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            printf("\nEnter the no of Employees:");
            scanf("%d", &n);
            for (i = 1; i <= n; i++)

                insertAtEnd();
            break;

        case 2:
            displayStatus();
            break;
        case 3:
            doubleEndedQueueDemo();
            break;
        case 4:
            exit(0);
        default:
            printf("\nEnter the valid choice");
        }
    }
}
```

OUTPUT:

8.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

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node *lchild;
    struct node *rchild;
};
typedef struct node *NODE;
NODE root = NULL;

void insert()
{
    NODE newnode, prev, curr;
    newnode = (NODE)malloc(sizeof(struct node));
    printf("Enter the element to be inserted in tree\n");
    scanf("%d", &newnode->data);
    newnode->lchild = newnode->rchild = NULL;
    if (root == NULL)
    {
        root = newnode;
        return;
    }
    prev = NULL;
    curr = root;
    while (curr != NULL)
    {
        if (curr->data == newnode->data)
        {
            printf("Duplicate is not possible\n");
            free(newnode);
            return;
        }
        prev = curr;
        if (newnode->data < curr->data)
            curr = curr->lchild;
        else
            curr = curr->rchild;
    }
    if (newnode->data < prev->data)
        prev->lchild = newnode;
    else
        prev->rchild = newnode;
    return;
}

void search(NODE root)
{
    int key;
    NODE curr;
    printf("Enter the element to be searched \n");
    scanf("%d", &key);
    if (root == NULL)
    {
        printf("tree is empty\n");
        return;
    }
    curr = root;
    while (curr != NULL)
    {
        if (curr->data == key)
        {
            printf("Number found\n");
            return;
        }
        if (key < curr->data)
            curr = curr->lchild;
        else
            curr = curr->rchild;
    }
    printf("Number not found\n");
    return;
}

void inorder(NODE root)
{
    if (root != NULL)
    {
        inorder(root->lchild);
        printf("%4d", root->data);
        inorder(root->rchild);
    }
    return;
}


void preorder(NODE root)
{
    if (root != NULL)
    {
        printf("%4d", root->data);
        preorder(root->lchild);
        preorder(root->rchild);
    }
    return;
}
void postorder(NODE root)
{
    if (root != NULL)
    {
        postorder(root->lchild);
        postorder(root->rchild);
        printf("%4d", root->data);
    }
    return;
}

void display(NODE root, int level)
{
    int i;
    if (root)
    {
        display(root->rchild, level + 1);
        printf("\n");
        for (i = 0; i < level; i++)
            printf("\t");
        printf("%d", root->data);
        display(root->lchild, level + 1);
    }
    return;
}

void main()
{
    int ch, i, n, item;
    NODE res;
    for (;;)
    {
        printf("\n**TREE OPERATIONS ARE**\n");
printf (" 1:Insert node\n 2:Inorder traversal\n 3:Preorder traversal\n
4:Postorder traversal\n5:Display\n 6:Search\n 7:Exit");
printf("\n*********\n"); printf("\nEnter your choice:\n");
scanf("%d",&ch);
switch(ch)
{
        case 1:
            printf("Enter the no of elements\n");
            scanf("%d", &n);
            for (i = 0; i < n; i++)
                insert();
            break;
        case 2:
            if (root == NULL)
                printf("Empty tree\n");
            else
            {
                printf("Inorder traversal\n");
                inorder(root);
            }
            break;
        case 3:
            if (root == NULL)
                printf("Empty tree\n");
            else
            {
                printf("preorder traversal\n");
                preorder(root);
            }
            break;
        case 4:
            if (root == NULL)
                printf("Emtpy tree\n");
            else
            {
                printf("Postorder traversal\n");
                postorder(root);
            }
            break;
        case 5:
            if (root == NULL)
                printf("Empty tree\n");
            else
                printf("The tree is:\n");
            display(root, 1);
            break;
        case 6:
            search(root);
            break;
        case 7:
            exit(0);
        default:
            printf("Invalid choice\n");
}
    }
}
```

OUTPUT:

9.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 methods.

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

int a[50][50], n, visited[50];
int q[20], front = -1,rear = -1;
int s[20], top = -1, count=0;

void bfs(int v)
{
         int i, cur;
         visited[v] = 1;
         q[++rear] = v;
         while(front!=rear)
        {
                cur = q[++front];
                for(i=1;i<=n;i++)
               {
                        if((a[cur][i]==1)&&(visited[i]==0))
                       {
                                 q[++rear] = i;
                                  visited[i] = 1;
                                  printf("%d ", i);
                        }
               }
        }
}

void dfs(int v)
{
         int i;
         visited[v]=1;
         s[++top] = v;
         for(i=1;i<=n;i++)
        {
                 if(a[v][i] == 1&& visited[i] == 0 )
                 {
                           printf("%d ", i);
                           dfs(i);
                  }
         }
}

int main()
{

         int ch, start, i,j;
         printf("\nEnter the number of vertices in graph:  ");
         scanf("%d",&n);
         printf("\nEnter the adjacency matrix:\n");
```

```
          for(i=1; i<=n; i++)
          {
                 for(j=1;j<=n;j++)
                        scanf("%d",&a[i][j]);
           }

     for(i=1;i<=n;i++)
            visited[i]=0;
     printf("\nEnter the starting vertex: ");
     scanf("%d",&start);

           printf("\n==>1. BFS: Print all nodes reachable from a given
starting node");
           printf("\n==>2. DFS: Print all nodes reachable from a given
starting node");
        printf("\n==>3:Exit");

        printf("\nEnter your choice: ");
        scanf("%d", &ch);

        switch(ch)
        {
              case 1: printf("\nNodes reachable from starting vertex %d
are: ", start);
                    bfs(start);
                    for(i=1;i<=n;i++)
                    {
                        if(visited[i]==0)
                             printf("\nThe vertex that is not reachable
is %d" ,i);
                    }
                    break;


              case 2: printf("\nNodes reachable from starting vertex %d
are:\n",start);
                    dfs(start);
                    break;
           case 3: exit(0);
           default: printf("\nPlease enter valid choice:");
        }


}
```

OUTPUT:

10.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 the Hash function H: K →L as H(K)=K mod m (remainder method) and implement a hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.

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

int key[20],n,m;
int *ht,index;
int count = 0;

void insert(int key)
{
            index = key % m;
            while(ht[index] != -1)
            {
                        index = (index+1)%m;
            }
            ht[index] = key;
            count++;
}

void display()
{
           int i;
           if(count == 0)
          {
                        printf("\nHash Table is empty");
                        return;
           }

           printf("\nHash Table contents are:\n ");
           for(i=0; i<m; i++)
                     printf("\n T[%d] --> %d ", i, ht[i]);
}


void main()
{
         int i;
         printf("\nEnter the number of employee  records (N) :   ");
         scanf("%d", &n);

         printf("\nEnter the two digit memory locations (m) for hash
table:   ");
         scanf("%d", &m);
         ht = (int *)malloc(m*sizeof(int));
         for(i=0; i<m; i++)
                     ht[i] = -1;

          printf("\nEnter the four digit key values (K) for N Employee
Records:\n  ");
         for(i=0; i<n; i++)
                    scanf("%d", &key[i]);

         for(i=0;i<n;i++)
        {
                   if(count == m)
                   {
       printf("\n~~~Hash table is full. Cannot insert the record %d
key~~~",i+1);
                        break;
                   }
                   insert(key[i]);
    }

            //Displaying Keys inserted into hash table
             display();
}
```

OUTPUT: