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). 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..

```c
#include <stdio.h>

#define MAX_DES_LENGTH 100

#define MAX_DAY_NAME_LENGTH 20
struct day{
    char dayName[MAX_DAY_NAME_LENGTH];
    int date;
    char dayDes[MAX_DES_LENGTH];
};
struct day weekcalender[7];
void read(){
    for(int i=0;i<7;++i){
        printf("ENTER THE DAY NAME OF DAY %d: \n",i+1);
        scanf("%s",weekcalender[i].dayName);
        printf("Enter the date for day %d :",i+1);
        scanf("%d",&weekcalender[i].date);
        printf("Enter Activity Description for day %d:",i+1);
        scanf("%[^\n]s",weekcalender[i].dayDes);
        }
}
void display(){
    printf("\n --calender--\n");
    for(int i=0;i<7;++i){
        printf("DAY %d :%s Date :%d
ActivityDescription:%s\n",i+1,weekcalender[i].dayName,weekcalender[i].date,weekcalender[i].dayDes );
    }
}

int main()
{
    printf("ENTER THE DETAILS OF EACH DAY\n");
    read();
    display;
    return 0;
}
```

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

```c
#include <stdio.h>

void read(char *s) {
    fgets(s, 100, stdin); // Use fgets for safer input
}

void strcopy(char *s1, const char *s2) {
    while ((*s1++ = *s2++)) {} // Use a simpler loop for string copying
}

int matchnreplace(char *str, const char *pat, const char *rep) {
    char ans[100] = {0}; // Initialize the array with zeros for safety
    int flag = 0;
    for (int c = 0; str[c] != '\0'; c++) {
        int i = 0, m = c, j = 0;
        while (str[m] == pat[i] && pat[i] != '\0') {
            i++;
            m++;
        }
        if (pat[i] == '\0') {
            flag = 1;
            for (int k = 0; rep[k] != '\0'; k++, j++)
                ans[j] = rep[k];
        } else {
            ans[j++] = str[c];
        }
    }
    ans[j] = '\0';
    strcopy(str, ans);
    return flag;
}

int main() {
    char str[100], pat[20], rep[20];
    printf("Enter the string: ");
    read(str);
    printf("Enter the pattern: ");
    read(pat);
    printf("Enter the string to be replaced: ");
    read(rep);
    if (matchnreplace(str, pat, rep))
        printf("The string after replacement: %s", str);
```

```
    else
        printf("The pattern is not found");
    return 0;
}
```


3. Develop a menu driven Program in C for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX) 1. Push an Element on to Stack 2. Pop an Element from Stack 3. Demonstrate how Stack can be used to check Palindrome 4. Demonstrate Overflow and Underflow situations on Stack 5. Display the status of Stack 6. Exit Support the program with a

ppropriate functions for each of the above operations charges

```c
#include <stdio.h>

#include <stdlib.h>

#define MAX 100

int stack[MAX];
int top = -1;

void push(int element) {
    if (top == MAX - 1) {
        printf("Stack Overflow\n");
        return;
    }
    stack[++top] = element;
}

int pop() {
    if (top == -1) {
        printf("Stack Underflow\n");
        return -1;
    }
    return stack[top--];
}

void display() {
    if (top == -1) {
        printf("Stack is empty\n");
        return;
    }
    printf("Stack elements are:\n");
    for (int i = top; i >= 0; i--) {
        printf("%d\n", stack[i]);
    }
}

int isPalindrome(char str[]) {
    int i = 0;
    int j = top;
    while (i < j) {
        if (str[i] != stack[j]) {
            return 0;
        }
        i++;
        j--;
    }
    return 1;
}

int main() {
    int choice, element;
    char str[MAX];

    while (1) {
        printf("\nStack Operations:\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Check Palindrome\n");
        printf("4. Display\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter element to push: ");
                scanf("%d", &element);
                push(element);
                break;
            case 2:
```

```
                element = pop();
                if (element != -1)
                    printf("Popped element: %d\n", element);
                break;
            case 3:
                printf("Enter a string: ");
                scanf("%s", str);
                if (isPalindrome(str))
                    printf("The entered string is a palindrome.\n");
                else
                    printf("The entered string is not a palindrome.\n");
                break;
            case 4:
                display();
                break;
            case 5:
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }

    return 0;
```

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.

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

ನಮ್ಮ ವಿಟಿಯು

```c
#include <stdlib.h>

#include <string.h>
#include <ctype.h>

#define MAX 100

char stack[MAX];
int top = -1;

// Function to push an element onto the stack
void push(char c) {
    stack[++top] = c;
}

// Function to pop an element from the stack
char pop() {
    return stack[top--];
}

// Function to determine the precedence of an operator
int precedence(char op) {
    return op == '^' ? 3 : (op == '*' || op == '/' || op == '%') ? 2 : (op == '+' || op == '-') ? 1 : -1;
}

// Function to check if a character is an operand
int isOperand(char c) {
    return isalnum(c); // Returns true if the character is alphanumeric
}

// Function to convert infix expression to postfix expression
void infixToPostfix(char* infix, char* postfix) {
    int i = 0, j = 0;
    char c;
    while ((c = infix[i++]) != '\0') {
        if (isOperand(c)) // If the character is an operand, add it to the postfix expression
            postfix[j++] = c;
        else if (c == '(') // If the character is '(', push it onto the stack
            push(c);
        else if (c == ')') { // If the character is ')', pop all operators from the stack until '(' is encountered
            while (top != -1 && stack[top] != '(')
                postfix[j++] = pop();
            if (top != -1) pop(); // Remove '('
        } else { // If the character is an operator
            while (top != -1 && precedence(c) <= precedence(stack[top]) && stack[top] != '(')
                postfix[j++] = pop(); // Pop operators with higher or equal precedence from the stack and add them to
the postfix expression
            push(c); // Push the current operator onto the stack
        }
    }
    // Pop any remaining operators from the stack and add them to the postfix expression
    while (top != -1)
        postfix[j++] = pop();
    postfix[j] = '\0'; // Null terminate the postfix expression
}

int main() {
    char infix[MAX], postfix[MAX];
    printf("Enter infix expression: ");
    fgets(infix, MAX, stdin); // Read infix expression from user
    infixToPostfix(infix, postfix); // Convert infix to postfix
    printf("Postfix expression: %s\n", postfix); // Print postfix expression
    return 0;
}
```

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

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

#define MAX 50

typedef struct {
    int top;
    int items[MAX];
} Stack;

void push(Stack *s, int value) {
    s->items[++(s->top)] = value;
}

int pop(Stack *s) {
    return s->items[(s->top)--];
}

int evaluatePostfix(char* exp) {
    Stack s = { .top = -1 };
    int i = 0;
    while(exp[i]) {
        if (isdigit(exp[i])) {
            push(&s, exp[i] - '0');
        } else {
            int op2 = pop(&s);
            int op1 = pop(&s);
            switch(exp[i]) {
                case '+': push(&s, op1 + op2); break;
                case '-': push(&s, op1 - op2); break;
                case '*': push(&s, op1 * op2); break;
                case '/': push(&s, op1 / op2); break;
                case '%': push(&s, op1 % op2); break;
                case '^': push(&s, op1 ^ op2); break;
            }
        }
        i++;
    }
    return pop(&s);
}
```

```c
int main() {
    char exp[MAX];
    printf("Enter the postfix expression: ");
    scanf("%s", exp);
    printf("The result of evaluation is : %d\n", evaluatePostfix(exp));
    return 0;
}
```

TOWER OF HONOI

```c
#include <stdio.h>

void towerOfHanoi(int n, char from_rod, char to_rod, char aux_rod) {
    if (n == 1) {
        printf("Move disk 1 from rod %c to rod %c\n", from_rod, to_rod);
        return;
    }
    towerOfHanoi(n-1, from_rod, aux_rod, to_rod);
    printf("Move disk %d from rod %c to rod %c\n", n, from_rod, to_rod);
    towerOfHanoi(n-1, aux_rod, to_rod, from_rod);
}

int main() {
    int n;
    printf("Enter the number of disks: ");
    scanf("%d", &n);
    towerOfHanoi(n, 'A', 'C', 'B');
    return 0;
}
```

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.

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

#define MAX 5

char queue[MAX];
int front = -1, rear = -1;

int isFull() {
    return ((front == 0 && rear == MAX - 1) || (front == rear + 1));
}

int isEmpty() {
    return (front == -1);
}

void enqueue(char item) {
    if (isFull()) {
        printf("Queue is full\n");
        return;
    }
    if (front == -1)
        front = 0;
    rear = (rear + 1) % MAX;
    queue[rear] = item;
    printf("%c enqueued to queue\n", item);
}

char dequeue() {
    char item;
    if (isEmpty()) {
        printf("Queue is empty\n");
```

```
        exit(1);
    }
    item = queue[front];
    if (front == rear) {
        front = -1;
        rear = -1;
    } else {
        front = (front + 1) % MAX;
    }
    return item;
}

void display() {
    if (isEmpty()) {
        printf("Queue is empty\n");
        return;
    }
    int i = front;
    printf("Queue elements: ");
    do {
        printf("%c ", queue[i]);
        i = (i + 1) % MAX;
    } while (i != (rear + 1) % MAX);
    printf("\n");
}

int main() {
    int choice;
    char item;

    while (1) {
        printf("\nCircular Queue Operations\n");
        printf("1. Insert\n");
        printf("2. Delete\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the element to insert: ");
                scanf(" %c", &item);
                enqueue(item);
                break;
            case 2:
                printf("Deleted element: %c\n", isEmpty() ? ' ' : dequeue());
                break;
            case 3:
                display();
```

```
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice!\n");
        }
    }

    return 0;
}
```

8.develop a menu driven Program in C for the following operations on Doubly Linked List (DLL) of Employee Data with the fields: SSN, Name, Dept, Designation, Sal, PhNo
a. Create a DLL of N Employees Data by using end insertion.
 b. Display the status of DLL and count the number of nodes in it
 c. Perform Insertion and Deletion at End of DLL
 d. Perform Insertion and Deletion at Front of DLL
 e. Demonstrate how this DLL can be used as Double Ended Queue

```
#include <stdio.h>
#include <stdlib.h>
struct Employee {
    char SSN[10];
    char Name[10];
    char Branch[10];
    char Des[10];
    char sal[10];
    char phone[10];
};
typedef struct Employee EMP;
struct node {
    EMP info;
    struct node *lptr, *rptr;
};
typedef struct node* Node;
Node front = NULL, rear = NULL;
EMP GetRec() {
    EMP temp;
    printf("Enter the SSN: ");
    fflush(stdin);
    gets(temp.SSN);
    printf("Enter the Name:");
    fflush(stdin);
    gets(temp.Name);
    printf("Enter branch:");
    fflush(stdin);
    gets(temp.Branch);
```

```
    printf("Enter the designation: ");
    fflush(stdin);
    gets(temp.Des);
    printf("Enter sal:");
    fflush(stdin);
    gets(temp.sal);
    printf("Enter the phone number:");
    fflush(stdin);
    gets(temp.phone);
    return temp;
}
void DispRec(EMP temp) {
    printf("%s\t", temp.SSN);
    printf("%s\t", temp.Name);
    printf("%s\t", temp.Branch);
    printf("%s\t", temp.Des);
    printf("%s\t", temp.sal);
    printf("%s\n", temp.phone);
}
Node Create(EMP ele) {
    Node temp;
    temp = (Node)malloc(sizeof(struct node));
    temp->info = ele;
    temp->lptr = NULL;
    temp->rptr = NULL;

    return temp;
}
void Finsert(EMP ele) {
    Node temp = Create(ele);
    if (front == NULL) {
        front = temp;
        rear = temp;
    } else {
        temp->rptr = front;
        front->lptr = temp;
        front = temp;
    }
}
void Einsert(EMP ele) {
    Node temp = Create(ele);
    if (rear == NULL) {
        front = temp;
        rear = temp;
    } else {
        rear->rptr = temp;
        temp->lptr = rear;
        rear = temp;
    }
}
```

ನಮ್ಮ ವಿಟಿಯು

```
void Fdelete()

{
    Node temp = front;
    if (front == NULL)
        printf("\nList is empty");
    else {
        temp = front;
        printf("\nDeleted Employee Records \n");
        DispRec(temp->info);
        front = front->rptr;
        front->lptr = NULL;
        if (front == NULL) rear = NULL;
        free(temp);
    }
}
void Edelete() {
    Node temp = rear, t;
    if (rear == NULL)
        printf("\nList is empty");
    else if (rear->lptr == NULL) {
        printf("\nDeleted Employee Records \n");
        DispRec(rear->info);
        front = rear = NULL;
        free(temp);
    } else {
        rear = rear->lptr;
        rear->rptr = NULL;
        printf("\nDeleted Employee Records \n");

        DispRec(temp->info);
        free(temp);
    }
}
void display(Node front) {
    Node temp = front;
    if (temp == NULL)
        printf("\nList is empty\n");
    else {
        printf("\nEmployee Records list\n");
        while (temp != NULL) {
            DispRec(temp->info);
            temp = temp->rptr;
        }
    }
}
int main() {
    EMP ele;
    int flag = 1, ch;
    while (flag) {
```

```
        printf("\n Menu Implementation of Double Ended Queue using DLL\n");
        printf(
            "\n1 Insert Front \n2 Insert rear \n3 Delete Front \n4 Delete Rear "
            "\n5 Display\n");
        printf("6 Exit\n Enter the Choice");
        scanf("%d", &ch);
        switch (ch) {
            case 1:
                printf(
                    "\n Enter the Employee Record to Insert Front of Queue\n");
                ele = GetRec();
                Finsert(ele);
                break;
            case 2: printf("\n Enter the Employee Record to Insert Front of
Queue\n");
ele=GetRec();
Einsert(ele);
break;
case 3: Fdelete();
break;
case 4: Edelete();
break;
case 5: display(front);
break;
default: flag = 0;
        }
    }
}
```

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

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

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

struct Student {
    char USN[15];
    char Name[50];
    char Programme[50];
```

```
    int Sem;
    long int PhNo;
    struct Student* next;
};

void createStudent(struct Student** head) {
    struct Student* newStudent = (struct Student*)malloc(sizeof(struct
Student));
    scanf("%s %s %s %d %ld", newStudent->USN, newStudent->Name, newStudent-
>Programme, &newStudent->Sem, &newStudent->PhNo);
    newStudent->next = *head;
    *head = newStudent;
}

void displayStudents(struct Student* head) {
    struct Student* temp = head;
    while (temp) {
        printf("USN: %s, Name: %s, Programme: %s, Semester: %d, PhNo: %ld\n",
               temp->USN, temp->Name, temp->Programme, temp->Sem, temp->PhNo);
        temp = temp->next;
    }
}

int countStudents(struct Student* head) {
    int count = 0;
    struct Student* temp = head;
    while (temp) {
        count++;
        temp = temp->next;
    }
    return count;
}

void insertAtFront(struct Student** head) {
    createStudent(head);
}

void deleteAtFront(struct Student** head) {
    if (*head) {
        struct Student* temp = *head;
        *head = (*head)->next;
        free(temp);
    }
}

int main() {
    struct Student* head = NULL;
    int choice;

    do {
```

```
        printf("\n----- MENU -----\n");
        printf("1. Create a SLL of N Students Data\n");
        printf("2. Display SLL status\n");
        printf("3. Insert at Front\n");
        printf("4. Delete at Front (Demonstration of stack)\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createStudent(&head);
                break;
            case 2:
                displayStudents(head);
                printf("Total students: %d\n", countStudents(head));
                break;
            case 3:
                insertAtFront(&head);
                break;
            case 4:
                deleteAtFront(&head);
                break;
            case 5:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Please enter a number between 1 and
5.\n");
        }
    } while (choice != 5);

    return 0;
}
```

```
9. Develop a Program in C for the following operationson Singly Circular
Linked List (SCLL) with header nodes
 a. Represent and Evaluate a Polynomial P(x,y,z) = 6x2y2z-4yz5+3x3yz+2xy5z-
2xyz3
 b. Find the sum of two polynomials POLY1(x,y,z) and POLY2(x,y,z) and store
the result in POLYSUM(x,y,z) Support the program with appropriate functions
for each of the above operations




#include <stdio.h>
```

ನಮ್ಮ ವಿಟಿಯು

```c
#include <stdlib.h>

struct Term {
    int coeff, x_exp, y_exp, z_exp;
    struct Term* next;
};

void insert(struct Term** poly, int coeff, int x_exp, int y_exp, int z_exp) {
    struct Term* newTerm = malloc(sizeof(struct Term));
    newTerm->coeff = coeff;
    newTerm->x_exp = x_exp;
    newTerm->y_exp = y_exp;
    newTerm->z_exp = z_exp;
    newTerm->next = *poly;
    *poly = newTerm;
}

void display(struct Term* poly) {
    if (!poly) {
        printf("Empty polynomial.\n");
        return;
    }
    while (poly) {
        printf("%+dx^%dy^%dz^%d ", poly->coeff, poly->x_exp, poly->y_exp, poly->z_exp);
        poly = poly->next;
    }
    printf("\n");
}

void evaluate(struct Term* poly) {
    int x = 2, y = 3, z = 4, result = 0;
    while (poly) {
        result += poly->coeff * (1 << poly->x_exp) * (1 << poly->y_exp) * (1 << poly->z_exp);
        poly = poly->next;
    }
    printf("%d\n", result);
}

void add(struct Term* poly1, struct Term* poly2, struct Term** result) {
    while (poly1) {
        insert(result, poly1->coeff, poly1->x_exp, poly1->y_exp, poly1->z_exp);
        poly1 = poly1->next;
    }
    while (poly2) {
        insert(result, poly2->coeff, poly2->x_exp, poly2->y_exp, poly2->z_exp);
        poly2 = poly2->next;
    }
}
```

```c
int main() {
    struct Term* POLY1 = NULL;
    struct Term* POLY2 = NULL;
    struct Term* POLYSUM = NULL;

    insert(&POLY1, 6, 2, 2, 1);
    insert(&POLY1, -4, 0, 1, 5);
    insert(&POLY1, 3, 3, 1, 1);

    insert(&POLY2, 2, 1, 5, 1);
    insert(&POLY2, -2, 1, 1, 3);

    printf("POLY1(x,y,z) = ");
    display(POLY1);
    printf("POLY2(x,y,z) = ");
    display(POLY2);

    printf("Evaluating POLY1(x=2, y=3, z=4): ");
    evaluate(POLY1);
    printf("Evaluating POLY2(x=2, y=3, z=4): ");
    evaluate(POLY2);

    add(POLY1, POLY2, &POLYSUM);
    printf("POLYSUM(x,y,z) = ");
    display(POLYSUM);

    return 0;
}
```

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

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

struct Node {
    int data;
    struct Node *left, *right;
};
```

ನಮ್ಮ ವಿಟಿಯು

```c
struct Node* newNode(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->left = node->right = NULL;
    return node;
}

struct Node* insert(struct Node* root, int data) {
    if (root == NULL) return newNode(data);
    if (data < root->data) root->left = insert(root->left, data);
    else if (data > root->data) root->right = insert(root->right, data);
    return root;
}

void inorder(struct Node* root) {
    if (root == NULL) return;
    inorder(root->left);
    printf("%d ", root->data);
    inorder(root->right);
}

void preorder(struct Node* root) {
    if (root == NULL) return;
    printf("%d ", root->data);
    preorder(root->left);
    preorder(root->right);
}

void postorder(struct Node* root) {
    if (root == NULL) return;
    postorder(root->left);
    postorder(root->right);
    printf("%d ", root->data);
}

struct Node* search(struct Node* root, int key) {
    if (root == NULL || root->data == key) return root;
    if (root->data < key) return search(root->right, key);
    return search(root->left, key);
}

int main() {
    struct Node* root = NULL;
    int choice, key;
    int elements[] = {6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2};
    int n = sizeof(elements) / sizeof(elements[0]);
    for (int i = 0; i < n; i++) root = insert(root, elements[i]);

    do {
        printf("\n----- MENU -----\n");
```

```
        printf("1. Inorder Traversal\n");
        printf("2. Preorder Traversal\n");
        printf("3. Postorder Traversal\n");
        printf("4. Search Element\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: printf("Inorder Traversal: "); inorder(root); printf("\n");
break;
            case 2: printf("Preorder Traversal: "); preorder(root);
printf("\n"); break;
            case 3: printf("Postorder Traversal: "); postorder(root);
printf("\n"); break;
            case 4: printf("Enter element to search: "); scanf("%d", &key);
                    if (search(root, key)) printf("Element found in the
BST.\n");
                    else printf("Element not found in the BST.\n");
                    break;
            case 5: printf("Exiting...\n"); break;
            default: printf("Invalid choice! Please enter a number between 1 and
5.\n");
        }
    } while (choice != 5);
    return 0;
}
```

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

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

#define MAX_CITIES 100
```

ನಮ್ಮ ವಿಟಿಯು

```
struct Graph {
    int vertices;
    bool adj[MAX_CITIES][MAX_CITIES];
};

void initGraph(struct Graph *graph) {
    graph->vertices = 0;
    for (int i = 0; i < MAX_CITIES; ++i)
        for (int j = 0; j < MAX_CITIES; ++j)
            graph->adj[i][j] = false;
}

void addEdge(struct Graph *graph, int src, int dest) {
    graph->adj[src][dest] = true;
}

// Depth First Search
void DFS(struct Graph *graph, int start, bool visited[]) {
    visited[start] = true;
    printf("%d ", start);
    for (int i = 0; i < graph->vertices; ++i)
        if (graph->adj[start][i] && !visited[i])
            DFS(graph, i, visited);
}

// Breadth First Search
void BFS(struct Graph *graph, int start) {
    bool visited[MAX_CITIES] = {false};
    int queue[MAX_CITIES], front = 0, rear = 0;

    visited[start] = true;
    queue[rear++] = start;

    while (front != rear) {
        int current = queue[front++];
        printf("%d ", current);

        for (int i = 0; i < graph->vertices; ++i)
            if (graph->adj[current][i] && !visited[i]) {
                visited[i] = true;
                queue[rear++] = i;
            }
    }
}

int main() {
    struct Graph graph;
    initGraph(&graph);

    int cities, edges, src, dest;
```

```
    printf("Enter number of cities: ");
    scanf("%d", &cities);
    graph.vertices = cities;

    printf("Enter number of edges: ");
    scanf("%d", &edges);

    printf("Enter edges (source destination):\n");
    for (int i = 0; i < edges; ++i) {
        scanf("%d %d", &src, &dest);
        addEdge(&graph, src, dest);
    }

    int start;
    printf("Enter starting city: ");
    scanf("%d", &start);

    printf("DFS traversal: ");
    bool visited[MAX_CITIES] = {false};
    DFS(&graph, start, visited);

    printf("\nBFS traversal: ");
    BFS(&graph, start);

    return 0;
}
```

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

```c
#include <stdio.h>
#include <stdlib.h>
int key[20], n, m;
int *ht, indX;
int count = 0;
void insert(int key) {
    indX = key % m;
    while (ht[indX] != -1) {
        indX = (indX + 1) % m;
    }
    ht[indX] = 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();
}
```