p1_text_editor_undo_redo.c

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

#define N 10

char text[N];
char redoStack[N];
int topText = -1;
int topRedo = -1;

void pushText(char ch)
{
    if (topText < N - 1)
    {
        text[++topText] = ch;
    }
    else
    {
        printf("Text buffer full\n");
    }
}

char popText()
{
    if (topText >= 0)
    {
        return text[topText--];
    }
    return '\0';
}

void pushRedo(char ch)
{
    if (topRedo < N - 1)
    {
        redoStack[++topRedo] = ch;
    }
    else
    {
        printf("Redo stack overflow\n");
    }
}

char popRedo()
{
    if (topRedo >= 0)
    {
        return redoStack[topRedo--];
    }
    return '\0';
}

void displayText()
{
    printf("Current Text: ");
    for (int i = 0; i <= topText; i++)
    {
        printf("%c", text[i]);
    }
    printf("\n");
}

int main()
{
    char ch;
    int choice;
```

```
    printf("---- Menu Choice ---- ");

    while (1)
    {
        printf("\n1. Push\n2. Undo\n3. Redo\n4. Display\n5. Exit\nEnter choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
            case 1:
                printf("Enter character to push: ");
                scanf(" %c", &ch);
                pushText(ch);
                topRedo = -1;  // Clear redo stack
                break;

            case 2:
                if (topText >= 0)
                {
                    char undoChar = popText();
                    pushRedo(undoChar);
                    printf("Undo: Removed %c\n", undoChar);
                }
                else
                {
                    printf("Nothing to undo.\n");
                }
                break;

            case 3:
                if (topRedo >= 0)
                {
                    char redoChar = popRedo();
                    pushText(redoChar);
                    printf("Redo: Restored %c\n", redoChar);
                }
                else
                {
                    printf("Nothing to redo.\n");
                }
                break;

            case 4:
                displayText();
                break;

            case 5:
                printf("Exited");
                exit(0);

            default:
                printf("Invalid choice.\n");
        }
    }
}
```

p2_queue_using_two_stacks.c

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

#define MAX 50

int stack1[MAX], stack2[MAX];
int top1 = -1, top2 = -1;
```

```
void push1(int val) {
    if (top1 == MAX - 1) {
        printf("Stack1 Overflow\n");
        return;
    }
    stack1[++top1] = val;
}

void push2(int val) {
    if (top2 == MAX - 1) {
        printf("Stack2 Overflow\n");
        return;
    }
    stack2[++top2] = val;
}

int pop1() {
    if (top1 == -1) return -1;
    return stack1[top1--];
}

int pop2() {
    if (top2 == -1) return -1;
    return stack2[top2--];
}

void enqueue(int val) {
    push1(val);
    printf("Enqueued: %d\n", val);
}

int dequeue() {
    if (top2 == -1) {
        while (top1 != -1) {
            push2(pop1());
        }
    }

    int val = pop2();
    if (val == -1) {
        printf("Queue is empty\n");
        return -1;
    }

    printf("Dequeued: %d\n", val);
    return val;
}

void displayQueue() {
    if (top1 == -1 && top2 == -1) {
        printf("Queue is empty\n");
        return;
    }

    printf("Queue elements: ");

    // First print stack2 (front of queue)
    for (int i = top2; i >= 0; i--) {
        printf("%d ", stack2[i]);
    }

    // Then print stack1 (back of queue)
    for (int i = 0; i <= top1; i++) {
        printf("%d ", stack1[i]);
    }

    printf("\n");
}

int main() {
```

```
    int ch, val;

    while (1) {
        printf("\n1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &ch);

        switch (ch) {
        case 1:
            printf("Enter value to enqueue: ");
            scanf("%d", &val);
            enqueue(val);
            break;

        case 2:
            dequeue();
            break;

        case 3:
            displayQueue();
            break;

        case 4:
            exit(0);

        default:
            printf("Invalid choice. Try again.\n");
        }
    }

    return 0;
}
```

p3_bill_management_linked_list.c

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int data ;
    struct node *next;
};
struct node *head,*temp;

// Function to add a new bill at the end
void addBill(int amount) {
    struct node *newNode = (struct node*)malloc(sizeof(struct node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        return;
    }
    newNode->data = amount;
    newNode->next = NULL;

    if (head == NULL) {
        head = newNode;
    } else {
        struct node *curr = head;
        while (curr->next != NULL)
            curr = curr->next;
        curr->next = newNode;
    }

    printf("Bill of Rs %d added successfully.\n", amount);
}
```

```
//Display Data
void display()
{
    if(head==0)
    {
        printf("No bills recorded yet \n");
    }
    temp=head;
    printf("Bills of the day :");
    while(temp!=0)
    {
        printf("%d rs ",temp->data);
        temp=temp->next;
    }
}

//insertion removed; using addBill instead

void total_sales() {
    if (head == NULL) {
        printf("No bills recorded yet.\n");
        return;
    }
    int total = 0;
    temp = head;
    while (temp != NULL) {
        total=total+temp->data;
        temp = temp->next;
    }
    printf("Total Sales of the day is %d rs\n", total);
}

void max_min_bill() {
    if (head == NULL) {
        printf("No bills recorded yet.\n");
        return;
    }
    temp = head;
    int max = head->data;
    int min = head->data;

    while (temp != NULL) {
        if (temp->data > max) max = temp->data;
        if (temp->data < min) min = temp->data;
        temp = temp->next;
    }

    printf("Maximum Bill is %d rs\n", max);
    printf("Minimum Bill is %d rs\n", min);
}
int main()
{
    int choice;
    while (choice != 5)
     {
        printf("\nCoffee Shop choices\n ");
        printf("1. Insert Bill of Customer\n");
        printf("2. Display all bills\n");
        printf("3. Display total sales\n");
        printf("4. Display maximum and minimum bill\n");
        printf("5. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                {
                    int amount;
                    printf("Enter bill amount : ");
                    if (scanf("%d", &amount) == 1) {
```

```
                        addBill(amount);
                    } else {
                        printf("Invalid amount input.\n");
                        // clear input buffer in a simple way
                        int c; while ((c = getchar()) != '\n' && c != EOF) {}
                    }
                }
                break;
            case 2:
                display();
                break;
            case 3:
                total_sales();
                break;
            case 4:
                max_min_bill();
                break;
            case 5:
                printf("Exit\n");
                break;
            default: printf("Invalid choice\n");
        }
    }
}
```

p4_music_playlist_circular_list.c

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

#define SIZE 100

// Structure to represent a Song (Node)
typedef struct Song {
    char name[SIZE];
    struct Song* next;
} Song;

Song* head = NULL;

// Function to create a new song node
Song* createSong(char* name) {
    Song* newSong = (Song*)malloc(sizeof(Song));
    strcpy(newSong->name, name);
    newSong->next = NULL;
    return newSong;
}

// 1. Insert a song
void insertSong(char* name) {
    Song* newSong = createSong(name);
    if (head == NULL) {
        head = newSong;
        newSong->next = head;
    } else {
        Song* temp = head;
        while (temp->next != head)
            temp = temp->next;
        temp->next = newSong;
        newSong->next = head;
    }
    printf("Song '%s' inserted.\n", name);
}
// 2. Delete a song
```

```
void deleteSong(char* name) {
    if (head == NULL) {
        printf("Playlist is empty.\n");
        return;
    }

    Song *current = head, *prev = NULL;

    // If head node itself holds the song to be deleted
    if (strcmp(head->name, name) == 0) {
        if (head->next == head) {
            free(head);
            head = NULL;
            printf("Song '%s' deleted.\n", name);
            return;
        } else {
            Song* temp = head;
            while (temp->next != head)
               {
                   temp = temp->next;
               }
            temp->next = head->next;
            Song* toDelete = head;
            head = head->next;
            free(toDelete);
            printf("Song '%s' deleted.\n", name);
            return;
        }
    }

    do {
        prev = current;
        current = current->next;
        if (strcmp(current->name, name) == 0) {
            prev->next = current->next;
            free(current);
            printf("Song '%s' deleted.\n", name);
            return;
        }
    } while (current != head);

    printf("Song '%s' not found in the playlist.\n", name);
}

// 3. Play songs (traversal)
void playSongs(int count) {
    if (head == NULL) {
        printf("Playlist is empty.\n");
        return;
    }
    Song* temp = head;
    printf("Playing %d songs (looping):\n", count);
    for (int i = 0; i < count; i++) {
        printf("Now Playing: %s\n", temp->name);
        temp = temp->next;
    }
}

// 4. Search for a song
void searchSong(char* name) {
    if (head == NULL) {
        printf("Playlist is empty.\n");
        return;
    }

    Song* temp = head;
    int pos = 1;
    do {
        if (strcmp(temp->name, name) == 0) {
            printf("Song '%s' found at position %d.\n", name, pos);
```

```
            return;
        }
        temp = temp->next;
        pos++;
    } while (temp != head);

    printf("Song '%s' not found in the playlist.\n", name);
}

// 5. Display playlist
void displayPlaylist() {
    if (head == NULL) {
        printf("Playlist is empty.\n");
        return;
    }

    Song* temp = head;
    printf("Current Playlist:\n");
    int index = 1;
    do {
        printf("%d. %s\n", index++, temp->name);
        temp = temp->next;
    } while (temp != head);
}

// 6. Count songs
int countSongs() {
    if (head == NULL)
        return 0;
    int count = 0;
    Song* temp = head;
    do {
        count++;
        temp = temp->next;
    } while (temp != head);
    return count;
}

// Main function with switch case
int main() {
    int choice, playCount;
    char name[SIZE];

    while (1) {
        printf("\n--- Music Playlist Menu ---\n");
        printf("1. Insert Song\n");
        printf("2. Delete Song\n");
        printf("3. Play Songs (Looped)\n");
        printf("4. Search Song\n");
        printf("5. Display Playlist\n");
        printf("6. Count Songs\n");
        printf("0. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        getchar(); // Consume newline

        switch (choice) {
            case 1:
                printf("Enter song name to insert: ");
                fgets(name, SIZE, stdin);
                name[strcspn(name, "\n")] = '\0'; // remove newline
                insertSong(name);
                break;

            case 2:
                printf("Enter song name to delete: ");
                fgets(name, SIZE, stdin);
                name[strcspn(name, "\n")] = '\0';
                deleteSong(name);
                break;
```

```
            case 3:
                printf("Enter how many songs you want to play: ");
                scanf("%d", &playCount);
                playSongs(playCount);
                break;

            case 4:
                printf("Enter song name to search: ");
                fgets(name, SIZE, stdin);
                name[strcspn(name, "\n")] = '\0';
                searchSong(name);
                break;

            case 5:
                displayPlaylist();
                break;

            case 6:
                printf("Total number of songs: %d\n", countSongs());
                break;

            case 0:
                printf("Exiting playlist program. Goodbye!\n");
                exit(0);

            default:
                printf("Invalid choice. Try again.\n");
        }
    }

    return 0;
}
```

p5_hash_table_linear_probing.c

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

int hash(int key) {
    return key % SIZE;
}

void insert(int hashTable[], int key) {
    int idx = hash(key);

    int first = idx;
    while (hashTable[idx] != -1) {
        idx = (idx + 1) % SIZE;
        if (idx == first) {
            printf("Hash Table is full, cannot insert %d\n", key);
            return;
        }
    }
    hashTable[idx] = key;
}

void display(int hashTable[]) {
    printf("\nHash Table:\n");
    for (int i = 0; i < SIZE; i++) {
        printf("Index %d : %d\n", i, hashTable[i]);
    }
}

int main() {
    int hashTable[SIZE];
```

```
    int choice, key;


    for (int i = 0; i < SIZE; i++) {
        hashTable[i] = -1;
    }

    while (1) {
        printf("\n--- Hash Table Menu ---\n");
        printf("1. Insert Key\n");
        printf("2. Display Hash Table\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter key to insert: ");
                scanf("%d", &key);
                insert(hashTable, key);
                break;

            case 2:
                display(hashTable);
                break;

            case 3:
                printf("Exiting program..\n");
                return 0;

            default:
                printf("Invalid choice!\n");
        }
    }
}
```

p6_employee_salary_bst.c

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

struct node {
    char name[50];
    int salary;
    struct node *left, *right;
};

// create a new node
struct node* createNode(char name[], int salary) {
    struct node* newNode = (struct node*)malloc(sizeof(struct node));
    strcpy(newNode->name, name);
    newNode->salary = salary;
    newNode->left = newNode->right = NULL;
    return newNode;
}

struct node* insert(struct node* root, char name[], int salary) {
    if (root == NULL){
        return createNode(name, salary);
    }
    else if (salary < root->salary){
        root->left = insert(root->left, name, salary);
    }
    else{
```

```
         root->right = insert(root->right, name, salary);
    }
    return root;
}

// display employees
void displayInorder(struct node* root) {
    if (root != NULL) {
        displayInorder(root->left);
        printf("%s\t%d\n", root->name, root->salary);
        displayInorder(root->right);
    }
}

// find min salary
struct node* minSalary(struct node* root) {
    if (root == NULL || root->left == NULL)
        return root;
    return minSalary(root->left);
}

// find max salary
struct node* maxSalary(struct node* root) {
    if (root == NULL || root->right == NULL)
        return root;
    return maxSalary(root->right);
}

// total salary expenses
int totalSalary(struct node* root) {
    if (root == NULL)
        return 0;
    return root->salary + totalSalary(root->left) + totalSalary(root->right);
}

int main() {
    struct node* root = NULL;
    int choice, salary;
    char name[50];

    while (1) {
        printf("\n\n----- Employee Database Menu -----\n");
        printf("1. Add Employee\n");
        printf("2. Display Employees (Sorted by Salary)\n");
        printf("3. Show Employee with Minimum Salary\n");
        printf("4. Show Employee with Maximum Salary\n");
        printf("5. Show Total Salary Expenses\n");
        printf("6. Exit\n");
        printf("---------------------------------\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter Employee Name: ");
                scanf("%s", name);
                printf("Enter Employee Salary: ");
                scanf("%d", &salary);
                root = insert(root, name, salary);
                printf("Employee Added!\n");
                break;

            case 2:
                if (root == NULL)
                    printf("No employees in database.\n");
                else {
                    printf("\nEmployees (Sorted by Salary):\n");
                    displayInorder(root);
                }
                break;
```

```
            case 3:
                if (root == NULL)
                    printf("No employees in database.\n");
                else {
                    struct node* min = minSalary(root);
                    printf("\nMinimum Salary: %s (%d)\n", min->name, min->salary);
                }
                break;

            case 4:
                if (root == NULL)
                    printf("No employees in database.\n");
                else {
                    struct node* max = maxSalary(root);
                    printf("\nMaximum Salary: %s (%d)\n", max->name, max->salary);
                }
                break;

            case 5:
                if (root == NULL)
                    printf("No employees in database.\n");
                else
                    printf("\nTotal Salary Expenses: %d\n", totalSalary(root));
                break;

            case 6:
                printf("Exiting Program...\n");
                exit(0);

            default:
                printf("Invalid Choice! Try again.\n");
        }
    }
    return 0;
}
```

p7_dictionary_trie.c

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

#define ALPHABET_SIZE 26
#define MAX_WORD_LEN 100

// Trie Node Structure
typedef struct TrieNode {
    struct TrieNode *children[ALPHABET_SIZE];
    int isEndOfWord;
} TrieNode;

// Create a new node
TrieNode* createNode() {
    TrieNode *node = (TrieNode*)malloc(sizeof(TrieNode));
    node->isEndOfWord = 0;

    for (int i = 0; i < ALPHABET_SIZE; i++)
        node->children[i] = NULL;

    return node;
}

// Convert a character to index (NO <ctype.h> USED)
// Works only for lowercase a–z
int getIndex(char ch) {
```

```
    if (ch >= 'a' && ch <= 'z')
        return ch - 'a';
    else
        return -1;   // invalid character
}

// Insert a word into Trie
void insert(TrieNode *root, const char *word) {
    TrieNode *curr = root;

    for (int i = 0; word[i] != '\0'; i++) {
        int index = getIndex(word[i]);
        if (index == -1)
            continue;   // skip invalid characters

        if (curr->children[index] == NULL)
            curr->children[index] = createNode();

        curr = curr->children[index];
    }

    curr->isEndOfWord = 1;
}

// Search a word in Trie
int search(TrieNode *root, const char *word) {
    TrieNode *curr = root;

    for (int i = 0; word[i] != '\0'; i++) {
        int index = getIndex(word[i]);
        if (index == -1)
            continue;

        if (curr->children[index] == NULL)
            return 0;

        curr = curr->children[index];
    }

    return curr->isEndOfWord;
}

// Insert words from file
void readFromFileAndInsert(TrieNode *root, const char *filename) {
    FILE *fp = fopen(filename, "r");

    if (fp == NULL) {
        printf("File not found.\n");
        return;
    }

    char buffer[1000];
    fgets(buffer, sizeof(buffer), fp);
    fclose(fp);

    char *token = strtok(buffer, "\t");
    while (token != NULL) {
        insert(root, token);
        token = strtok(NULL, "\t");
    }

    printf("Words inserted from file successfully!\n");
}

// Insert from array
void readFromArrayAndInsert(TrieNode *root, char keys[][MAX_WORD_LEN], int n) {
    for (int i = 0; i < n; i++)
        insert(root, keys[i]);

    printf("Words inserted from array successfully!\n");
```

```
}

// Direct search in file without Trie
void searchArrayKeysInFile(char keys[][MAX_WORD_LEN], int n, const char *filename) {
    FILE *fp = fopen(filename, "r");

    if (fp == NULL) {
        printf("File not found.\n");
        return;
    }

    char buffer[1000];
    fgets(buffer, sizeof(buffer), fp);
    fclose(fp);

    for (int i = 0; i < n; i++) {
        if (strstr(buffer, keys[i]) != NULL)
            printf("'%s' found in file.\n", keys[i]);
        else
            printf("'%s' not found in file.\n", keys[i]);
    }
}

int main() {
    TrieNode *root = createNode();
    int choice;
    char input[MAX_WORD_LEN];

    // Sample words
    char keys[][MAX_WORD_LEN] = {
        "and","bat","ball","book","cot","cotton",
        "internet","interview","joy","job","king",
        "lion","man","mango","manage"
    };
    int keyCount = sizeof(keys) / sizeof(keys[0]);

    // Write array words into a file
    FILE *fp = fopen("keys.txt", "w");
    for (int i = 0; i < keyCount; i++)
        fprintf(fp, "%s\t", keys[i]);
    fclose(fp);

    printf("\n==== TRIE WORD SEARCH PROGRAM (Beginner-Friendly) ====\n");

    while (1) {
        printf("\nMenu:\n");
        printf("1. Read from file and search word\n");
        printf("2. Read from array and search word\n");
        printf("3. Search all array words in file (no trie)\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                readFromFileAndInsert(root, "keys.txt");
                printf("Enter word to search (lowercase only): ");
                scanf("%s", input);

                if (search(root, input))
                    printf("'%s' FOUND in Trie.\n", input);
                else
                    printf("'%s' NOT found in Trie.\n", input);
                break;

            case 2:
                readFromArrayAndInsert(root, keys, keyCount);
                printf("Enter word to search (lowercase only): ");
                scanf("%s", input);
                if (search(root, input))
```

```
                printf("'%s' FOUND in Trie.\n", input);
            else
                printf("'%s' NOT found in Trie.\n", input);
            break;

        case 3:
            searchArrayKeysInFile(keys, keyCount, "keys.txt");
            break;

        case 4:
            printf("Exiting... Goodbye!\n");
            exit(0);

        default:
            printf("Invalid choice. Try again.\n");
        }
    }

    return 0;
}
```

p8_network_critical_node_detection.c

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

#define MAX 100

int adj[MAX][MAX];  // Adjacency matrix
int visited[MAX];   // Visited array
int n;              // Number of nodes

// Standard DFS ignoring the skipped node
void dfs(int v, int skip) {
    visited[v] = 1;
    for (int i = 0; i < n; i++) {
        if (i != skip && adj[v][i] && !visited[i])
            dfs(i, skip);
    }
}

// Check if the graph is connected when node 'skip' is removed
int isConnected(int skip) {
    for (int i = 0; i < n; i++)
        visited[i] = 0;

    // Find a starting node that is not the skipped node
    int start = -1;
    for (int i = 0; i < n; i++) {
        if (i != skip) {
            start = i;
            break;
        }
    }

    if (start == -1)
        return 1; // Only one node? It's trivially connected

    dfs(start, skip);

    // Check if all nodes except skipped are visited
    for (int i = 0; i < n; i++) {
        if (i != skip && !visited[i])
            return 0; // Not connected
    }
```

```
    return 1; // Connected
}

int main() {
    printf("Enter number of nodes: ");
    scanf("%d", &n);

    printf("Enter adjacency matrix:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            scanf("%d", &adj[i][j]);

    printf("\nCritical node analysis:\n");
    for (int i = 0; i < n; i++) {
        if (!isConnected(i))
            printf("Node %d is critical. Its failure disconnects the network.\n", i);
        else
            printf("Node %d is not critical.\n", i);
    }

    return 0;
}
```

p9_graph_cycle_detection.c

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

#define MAX 100

int adj[MAX][MAX];
bool visited[MAX];
int V;

// Function for DFS to detect cycle
bool dfsCycle(int v, int parent) {
    visited[v] = true;

    for (int u = 0; u < V; u++) {
        if (adj[v][u]) { // if there is an edge
            if (!visited[u]) {
                if (dfsCycle(u, v))
                    return true;
            }
            else if (u != parent) {
                return true; // cycle found
            }
        }
    }
    return false;
}

// Function to check if graph has a cycle
bool isCyclic() {
    for (int i = 0; i < V; i++)
        visited[i] = false;

    for (int i = 0; i < V; i++) {
        if (!visited[i]) {
            if (dfsCycle(i, -1))
                return true;
        }
    }
```

```
    return false;
}

int main() {
    int choice;
    printf("----- CYCLE DETECTION IN GRAPH -----\n");
    printf("1. Read Graph from File\n");
    printf("2. Read Graph as Adjacency List\n");
    printf("3. Read Graph as Adjacency Matrix\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    // Initialize adjacency matrix
    memset(adj, 0, sizeof(adj));

    switch (choice) {
        case 1: { // Read graph from file
            FILE *file = fopen("graph.txt", "r");
            if (file == NULL) {
                printf("File not found!\n");
                return 0;
            }
            int E, u, v;
            fscanf(file, "%d %d", &V, &E);
            for (int i = 0; i < E; i++) {
                fscanf(file, "%d %d", &u, &v);
                adj[u][v] = 1;
                adj[v][u] = 1;
            }
            fclose(file);
            break;
        }

        case 2: { // Read from adjacency list
            printf("Enter number of vertices: ");
            scanf("%d", &V);
            for (int i = 0; i < V; i++) {
                printf("Enter adjacent vertices of %d (-1 to stop): ", i);
                int node;
                while (1) {
                    scanf("%d", &node);
                    if (node == -1)
                        break;
                    adj[i][node] = 1;
                    adj[node][i] = 1;
                }
            }
            break;
        }

        case 3: { // Read from adjacency matrix
            printf("Enter number of vertices: ");
            scanf("%d", &V);
            printf("Enter adjacency matrix:\n");
            for (int i = 0; i < V; i++) {
                for (int j = 0; j < V; j++) {
                    scanf("%d", &adj[i][j]);
                }
            }
            break;
        }

        default:
            printf("Invalid choice!\n");
            return 0;
    }

    if (isCyclic())
        printf("\nCycle Detected in Graph.\n");
    else
```

```
        printf("\nNo Cycle Found in Graph.\n");

    return 0;
}
```