Problem 6 :- To implement queue data structures and explore their applications.

Code :-

#include <stdio.h>

#define SIZE 5

int queue[SIZE];

int front = -1, rear = -1;

void enqueue(int value) {

if(rear == SIZE - 1)

printf("Queue is full\n");

else {

if(front == -1)

front = 0;

queue[++rear] = value;

}

}

int dequeue() {

if(front == -1 || front > rear)

printf("Queue is empty\n");

else

return queue[front++];

return -1;

}

int main() {

enqueue(10);

enqueue(20);

printf("Dequeued: %d\n", dequeue());

return 0;

} //KushSaraf

Output :- 

Problem 7 :- To implement singly linked lists and perform basic operations on them.

Code :-

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node \*next;

};

void insert(struct Node\*\* head, int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->next = \*head;

\*head = newNode;

}

void display(struct Node\* head) {

struct Node\* temp = head;

while(temp != NULL) {

printf("%d -> ", temp->data);

temp = temp->next;

}

printf("NULL\n");

}

int main() {

struct Node\* head = NULL;

insert(&head, 10);

insert(&head, 20);

display(head);

return 0;

} //KushSaraf

Output :-



Problem 8 :- To implement binary trees and perform different tree traversal techniques.

Code :-

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node \*left, \*right;

};

struct Node\* newNode(int data) {

struct Node\* node = (struct Node\*)malloc(sizeof(struct Node));

node->data = data;

node->left = node->right = NULL;

return node;

}

void inorder(struct Node\* root) {

if (root == NULL) return;

inorder(root->left);

printf("%d ", root->data);

inorder(root->right);

}

int main() {

struct Node\* root = newNode(1);

root->left = newNode(2);

root->right = newNode(3);

inorder(root);

return 0;

} //KushSaraf

Output :-



Problem 9 :- To implement binary search trees and perform insertion, deletion, and search operations.

Code :-

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node \*left, \*right;

};

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\* node, int data) {

if (node == NULL) return newNode(data);

if (data < node->data) node->left = insert(node->left, data);

else if (data > node->data) node->right = insert(node->right,data);

return node;

}

int main() {

struct Node\* root = NULL;

root = insert(root, 10);

root = insert(root, 5);

root = insert(root, 20);

printf("BST created with nodes.\n");

return 0;

} //KushSaraf

Output :-



Problem 10 :- To represent graphs using adjacency matrices and lists, and implement BFS and DFS traversal algorithms.

Code :-

#include <stdio.h>

#include <stdlib.h>

#define V 5

void addEdge(int adj[V][V], int src, int dest) {

adj[src][dest] = 1;

adj[dest][src] = 1;

}

void bfs(int adj[V][V], int start) {

int visited[V] = {0}, queue[V], front = 0, rear = 0;

visited[start] = 1;

queue[rear++] = start;

while (front != rear) {

int node = queue[front++];

printf("%d ", node);

for (int i = 0; i < V; i++) {

if (adj[node][i] && !visited[i]) {

visited[i] = 1;

queue[rear++] = i;

}

}

}

}

int main() {

int adj[V][V] = {0};

addEdge(adj, 0, 1);

addEdge(adj, 1, 2);

bfs(adj, 0);

return 0;

} //KushSaraf

Output :- 