#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define NAME\_LENGTH 50

// --------------------- Queue for Voters ---------------------

typedef struct Voter {

int id;

struct Voter\* next;

} Voter;

Voter \*front = NULL, \*rear = NULL;

void enqueue(int id) {

Voter\* newVoter = (Voter\*)malloc(sizeof(Voter));

newVoter->id = id;

newVoter->next = NULL;

if (rear == NULL)

front = rear = newVoter;

else {

rear->next = newVoter;

rear = newVoter;

}

printf("Voter with ID %d registered.\n", id);

}

int dequeue() {

if (front == NULL)

return -1;

int id = front->id;

Voter\* temp = front;

front = front->next;

if (front == NULL) rear = NULL;

free(temp);

return id;

}

// ------------------ Linked List for Candidates ------------------

typedef struct Candidate {

int id;

char name[NAME\_LENGTH];

int votes;

struct Candidate\* next;

} Candidate;

Candidate\* candidateHead = NULL;

void addCandidate(int id, char name[]) {

Candidate\* newNode = (Candidate\*)malloc(sizeof(Candidate));

newNode->id = id;

strcpy(newNode->name, name);

newNode->votes = 0;

newNode->next = candidateHead;

candidateHead = newNode;

}

void displayCandidates() {

Candidate\* temp = candidateHead;

printf("\n--- Candidates List ---\n");

while (temp != NULL) {

printf("ID: %d | Name: %s | Votes: %d\n", temp->id, temp->name, temp->votes);

temp = temp->next;

}

}

// ------------------ Tree for Vote Results ------------------

typedef struct VoteNode {

int candidateId;

int voteCount;

struct VoteNode\* left;

struct VoteNode\* right;

} VoteNode;

VoteNode\* root = NULL;

VoteNode\* insertVote(VoteNode\* node, int candidateId) {

if (node == NULL) {

VoteNode\* newNode = (VoteNode\*)malloc(sizeof(VoteNode));

newNode->candidateId = candidateId;

newNode->voteCount = 1;

newNode->left = newNode->right = NULL;

return newNode;

}

if (candidateId == node->candidateId)

node->voteCount++;

else if (candidateId < node->candidateId)

node->left = insertVote(node->left, candidateId);

else

node->right = insertVote(node->right, candidateId);

return node;

}

void displayResults(VoteNode\* node) {

if (node == NULL) return;

displayResults(node->left);

printf("Candidate ID %d got %d votes.\n", node->candidateId, node->voteCount);

displayResults(node->right);}

// ------------------ Vote Functionality ------------------

void castVote() {

if (front == NULL) {

printf("No voters in queue.\n");

return;

}

int voterId = dequeue();

int choice;

Candidate\* temp = candidateHead;

printf("\nVoter %d voting...\n", voterId);

displayCandidates();

printf("Enter candidate ID to vote: ");

scanf("%d", &choice);

while (temp != NULL) {

if (temp->id == choice) {

temp->votes++;

root = insertVote(root, choice);

printf("Vote cast successfully by voter %d.\n", voterId);

return;

}

temp = temp->next;

}

printf("Invalid candidate ID.\n");

}

// ------------------ Main Program ------------------

int main() {

int ch, id;

char name[NAME\_LENGTH];

printf("=== Digital Voting System ===\n");

while (1) {

printf("\n1. Add Candidate\n2. Register Voter\n3. Cast Vote\n4. Show Candidates\n5. Show Results\n6. Exit\nChoice: ");

scanf("%d", &ch);

switch (ch) {

case 1:

printf("Enter candidate ID: ");

scanf("%d", &id);

printf("Enter candidate name: ");

scanf(" %[^\n]s", name);

addCandidate(id, name);

break;

case 2:

printf("Enter voter ID: ");

scanf("%d", &id);

enqueue(id);

break;

case 3:

castVote();

break;

case 4:

displayCandidates();

break;

case 5:

printf("\n--- Voting Results ---\n");

displayResults(root);

break;

case 6:

printf("Exiting voting system.\n");

exit(0);

default:

printf("Invalid choice.\n");

}

}

return 0;

}

This project is a simple **Digital Voting System implemented in C** using fundamental **Data Structures**: **Queue**, **Linked List**, and **Binary Tree**.

**🔹 1. Voter Registration – Queue**

* Voters are registered using a **FIFO queue**.
* Each voter is added to the queue with a unique ID.
* When a voter casts their vote, they are **removed (dequeued)** from the front of the queue.

**🔹 2. Candidate Management – Linked List**

* Candidates are stored in a **singly linked list**.
* Each node contains the **candidate ID, name, and vote count**.
* New candidates can be added dynamically.

**🔹 3. Vote Casting and Tracking – Binary Tree**

* Votes are stored in a **binary search tree (BST)**.
* Each node represents a **candidate ID and total votes received**.
* Allows sorted display of results using **in-order traversal**.

**🔹 4. Main Features**

* Add new candidates.
* Register voters and manage them in queue.
* Voters vote for candidates by ID.
* Display candidate list and real-time vote results.

**🔹 Usage**

The program runs in a **menu-driven format**, allowing the user to interactively:

* Register voters
* Add candidates
* Cast votes
* Display candidates and voting results