

**UNIVERSITY OF PETROLEUM AND ENERGY STUDIES**

DEHRADUN, UTTRAKHAND



**SCHOOL OF COMPUTER SCIENCE**

DATA STRUCTURES AND ALGORITHMS LAB (CSEG1034\_1)

**DATA STRUCTURES AND ALGORITHMS PROJECT FILE  
(VOTING MANAGEMENT SYSTEM)**

(2023-24)

---

**Submitted To:**

**Dr. Virender Kadyan**

Associate Professor

School of Computer Science

**Submitted By:**

**Aditya Shrivastava**

B. Tech CSE (2<sup>nd</sup> Semester)

SAP ID :500124727

Enrolment No.: R2142231558

For accessing the source code, documentation and all the resources are available at **GitHub**



# PROJECT REPORT

## 1. Project Title:

Election Voting Management System

## 2. Abstract:

The Election Voting Management System is a CLI based application designed to manage and streamline the process of conducting elections. It aims to automate the traditional manual process, reducing the chances of errors and increasing the efficiency of the overall voting process. The system can handle various tasks such as voter registration, vote casting, and vote counting, ensuring a transparent and secure election process.

## 3. Problem Statement:

In traditional election processes, there are numerous challenges such as manual voter registration, vote casting, and vote counting. These processes are time-consuming, prone to human error, and lack transparency, which can lead to disputes over election results. The need for a secure, efficient, and transparent system for managing elections is evident. The Simple Election Voting Management System aims to address these issues by automating these processes, ensuring accuracy, efficiency, and transparency in elections.

## 4. Objective:

The objective of the Election Voting Management System is to automate and streamline the election process.

## 5. Methodology:

The methodology for developing the Election Voting Management System involves the following steps:

- Requirement Analysis: Identify and document the functional and non-functional requirements of the system. This includes understanding the election process, voter registration, vote casting, and vote counting procedures.

- **System Design:** Design the system architecture and database schema. This includes designing the user interface for voter registration and vote casting, and the algorithm for vote counting.
- **Implementation:** Write the code for the system in C programming language. This includes coding the user interface, database interactions, and the vote counting algorithm.
- **Testing:** Test the system to ensure it meets all the functional and non-functional requirements. This includes unit testing, integration testing, and system testing.
- **Deployment:** Deploy the system for use in an election. This includes setting up the system on the required hardware and providing training to the users.
- **Maintenance:** Monitor the system during the election to ensure it operates without any issues. This includes troubleshooting any issues that arise and making necessary updates to the system.

## CODE

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

typedef struct Voter {
    int id;
    struct Voter *next;
} Voter;

typedef struct {
    char name[100];
    int votes;
    Voter *voters_head;
} Candidate;

Candidate *candidates = NULL;
int num_candidates = 0;

void add_candidate(const char *name);
int find_candidate(const char *name);
void register_vote(int voter_id, const char *candidate_name);
void display_results();
void save_data();
void load_data();
void free_memory();
```

```

int main() {

    load_data(); // Load previously saved data

    int choice;

    char name[100];

    int voter_id;

    while (1) {

        printf("\n••••• Election Voting Management System •••••\n");

        printf("\n1. Add Candidate\n2. Cast Vote\n3. Display Results\n4. Save and Exit\nEnter
your choice: ");

        fgets(name, sizeof(name), stdin); // Flush stdin

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("Enter candidate name: ");

                scanf(" %[^\n]s", name); // Read string with spaces

                add_candidate(name);

                break;

            case 2:

                printf("Enter voter ID: ");

                scanf("%d", &voter_id);

                printf("Enter candidate name: ");

                scanf(" %[^\n]s", name); // Read string with spaces

                register_vote(voter_id, name);

                break;

            case 3:

                display_results();

                break;

            case 4:

                save_data();

```

```

        free_memory(); // Free the allocated memory
        return 0;
    default:
        printf("Invalid choice. Please try again.\n");
    }
}
return 0;
}

```

```

void add_candidate(const char *name) {
    if (find_candidate(name) != -1) {
        printf("Candidate '%s' already exists.\n", name);
        return;
    }
    candidates = realloc(candidates, (num_candidates + 1) * sizeof(Candidate));
    strcpy(candidates[num_candidates].name, name);
    candidates[num_candidates].votes = 0;
    candidates[num_candidates].voters_head = NULL;
    num_candidates++;
    printf("Candidate '%s' added successfully.\n", name);
}

```

```

int find_candidate(const char *name) {
    for (int i = 0; i < num_candidates; i++) {
        if (strcmp(candidates[i].name, name) == 0) {
            return i;
        }
    }
    return -1;
}

```

```
}
```

```
void register_vote(int voter_id, const char *candidate_name) {  
    int idx = find_candidate(candidate_name);  
    if (idx == -1) {  
        printf("Candidate not found.\n");  
        return;  
    }  
    Voter *current = candidates[idx].voters_head;  
    while (current) {  
        if (current->id == voter_id) {  
            printf("Voter %d has already voted.\n", voter_id);  
            return;  
        }  
        current = current->next;  
    }  
    Voter *newVoter = (Voter *)malloc(sizeof(Voter));  
    newVoter->id = voter_id;  
    newVoter->next = candidates[idx].voters_head;  
    candidates[idx].voters_head = newVoter;  
    candidates[idx].votes++;  
    printf("Vote registered for %s by voter %d.\n", candidate_name, voter_id);  
}
```

```
void display_results() {  
    printf("\nElection Results:\n");  
    for (int i = 0; i < num_candidates; i++) {  
        printf("%s: %d votes\n", candidates[i].name, candidates[i].votes);  
    }  
}
```



```
}
```

```
void save_data() {
```

```
    FILE *file = fopen("voting_data.txt", "w");
```

```
    if (!file) {
```

```
        printf("Failed to save data.\n");
```

```
        return;
```

```
    }
```

```
    for (int i = 0; i < num_candidates; i++) {
```

```
        fprintf(file, "%s %d\n", candidates[i].name, candidates[i].votes);
```

```
    }
```

```
    fclose(file);
```

```
    printf("Data saved successfully.\n");
```

```
}
```

```
void load_data() {
```

```
    FILE *file = fopen("voting_data.txt", "r");
```

```
    if (!file) return; // No data file, start fresh
```

```
    char name[100];
```

```
    int votes;
```

```
    while (fscanf(file, "%99s %d", name, &votes) == 2) {
```

```
        add_candidate(name);
```

```
        candidates[num_candidates - 1].votes = votes; // Set votes from loaded data
```

```
    }
```

```
    fclose(file);
```

```
}
```

```
void free_memory() {
```

```
    for (int i = 0; i < num_candidates; i++) {
```

```
Voter *current = candidates[i].voters_head;
while (current) {
    Voter *tmp = current;
    current = current->next;
    free(tmp);
}
free(candidates);
}
```

## OUTPUT

```
PS D:\DSA\C> & 'c:\Users\aditya\.vscode\extensions\Microsoft-MIEngine-In-r2l3mg1h.adc' '--stdout=M
'--pid=Microsoft-MIEngine-Pid-rkmmzzat.ph0' '--dbgE

..... Election Voting Management System .....

1. Add Candidate
2. Cast Vote
3. Display Results
4. Save and Exit
Enter your choice:
1
Enter candidate name: Aditya Shrivastava
Candidate 'Aditya Shrivastava ' added successfully.

..... Election Voting Management System .....

1. Add Candidate
2. Cast Vote
3. Display Results
4. Save and Exit
Enter your choice: 1
Enter candidate name: Anshuman Agrawal
Candidate 'Anshuman Agrawal ' added successfully.

..... Election Voting Management System .....

1. Add Candidate
2. Cast Vote
3. Display Results
4. Save and Exit
Enter your choice: 2
Enter voter ID: 500124727
Enter candidate name: Aditya Shrivastava
Vote registered for Aditya Shrivastava by voter 500124727.

..... Election Voting Management System .....

1. Add Candidate
2. Cast Vote
3. Display Results
4. Save and Exit
Enter your choice: 2
Enter voter ID: 500126680
Enter candidate name: Anshuman Agrawal
Vote registered for Anshuman Agrawal by voter 500126680.

..... Election Voting Management System .....

1. Add Candidate
2. Cast Vote
3. Display Results
4. Save and Exit
Enter your choice: 2
Enter voter ID: 500120680
Enter candidate name: Aditya Shrivastava
Vote registered for Aditya Shrivastava by voter 500120680.
```

```
..... Election Voting Management System .....
```


1. Add Candidate
  2. Cast Vote
  3. Display Results
  4. Save and Exit
- Enter your choice: 3

Election Results:  
Aditya Shrivastava : 2 votes  
Anshuman Agrawal : 1 votes

```
..... Election Voting Management System .....
```

1. Add Candidate
  2. Cast Vote
  3. Display Results
  4. Save and Exit
- Enter your choice: 4  
Data saved successfully.

## STORING DATA (FILE HANDLING)

```
Project >  voting_data.txt
1   Aditya Shrivastava  2
2   Anshuman Agrawal   1
3
```

# TEST CASES

## 1. Test `add\_candidate` Function

### Test Case 1: Add a new candidate.

- Input: Name = "ABC"
- Output: "Candidate 'ABC' added successfully."
- Purpose: Tests if a new candidate can be added.

### Test Case 2: Try adding the same candidate again.

- Input: Name = "ABC"
- Output: "Candidate 'ABC' already exists."
- Purpose: Checks if the system prevents duplicate candidates.

## 2. Test `find\_candidate` Function

### Test Case 3: Find an existing candidate.

- Input: Name = "ABC"
- Output: Returns index of "ABC"
- Purpose: Ensures that the function can locate an existing candidate.

### Test Case 4: Try to find a non-existing candidate.

- Input: Name = "XYZ"
- Output: Returns -1
- Purpose: Tests the function's ability to return -1 for non-existent candidates.

### 3. Test `register\_vote` Function

Test Case 5: Register a vote for an existing candidate.

- Input: Voter ID = 1, Candidate Name = "ABC"
- Output: "Vote registered for ABC by voter 1."
- Purpose: Verifies if a vote can be correctly registered.

Test Case 6: Register a vote from the same voter for the same candidate.

- Input: Voter ID = 1, Candidate Name = "ABC"
- Output: "Voter 1 has already voted."
- Purpose: Checks if the system prevents the same voter from voting multiple times for the same candidate.

Test Case 7: Try to vote for a non-existing candidate.

- Input: Voter ID = 2, Candidate Name = "XYZ"
- Output: "Candidate not found."
- Purpose: Ensures that votes cannot be cast for non-existing candidates.

### 4. Test `display\_results` Function

Test Case 8: Display results after some votes.

- Output: Proper listing of candidates and their vote counts.
- Purpose: To see if the system correctly displays the voting results.

## 5. Test `save\_data` and `load\_data` Functions

### Test Case 9: Save and load the current state.

- Action: Add candidates, register some votes, save data, reset the program, and load data.
- Output: After reload, the state should be restored accurately.
- Purpose: Tests persistence and data integrity across sessions.

## 6. Test Memory Management

### Test Case 10: Check for memory leaks and proper freeing of memory.

- Action: Run the program with memory debugging tools like Valgrind after performing all other tests.
- Output: No memory leaks reported.
- Purpose: Ensures that all allocated memory is freed and there are no memory leaks.

## COMMENTS