FINAL PROJECT REPORT – PROGRAMMING FUNDAMENTALS

University Name: FAST National University

Department: Department of Data Science

Course: Programming Fundamentals

Project Title: Election System

Submitted By: Muhammad Arqam (25K-2542) & Waleed Ahmed (25K-2517)

Submitted To: Mr. Jahanzaib (Course Instructor)

Semester: Fall 2025

Date: 20/11/2025

ABSTRACT

The following project represents the implementation of a console-based voting system using the C programming language. It allows administrators to securely add and delete candidates through password-protected access, while general users can input votes by name with a 13-digit unique ID. The program ensures fairness in that no duplicate voting is allowed through verification of ID and provides complete transparency in the form of candidate and vote count lists, as well as a detailed record of voting. The program also includes structured menus, input validation, and dynamic management of candidate and voter lists. Overall, the project shows the practical implementation of arrays, string handling, decision structures, and user-defined functions for the development of an electronic voting application that will be simple yet functional.

INTRODUCTION

The electronic voting process has emerged as an extremely important solution for conducting secure, transparent, and efficient voting in educational, organizational, or small environments. This project will consist of a simple console-based voting application coded in C to illustrate the basic components of an election. Candidates will be administered under secure access. Administrators will have secure access protected by a password, and voters will securely vote using a unique identifier. The program provides an example of how these programming concepts of arrays, loops, conditional statements, and functions can be applied to develop a solution to a real-world problem by implementing input validation, duplicate voting prevention, and tracking the votes in real time. Overall, this project's main purpose is to produce an easily manageable, fair-controlled, and accessible voting experience.

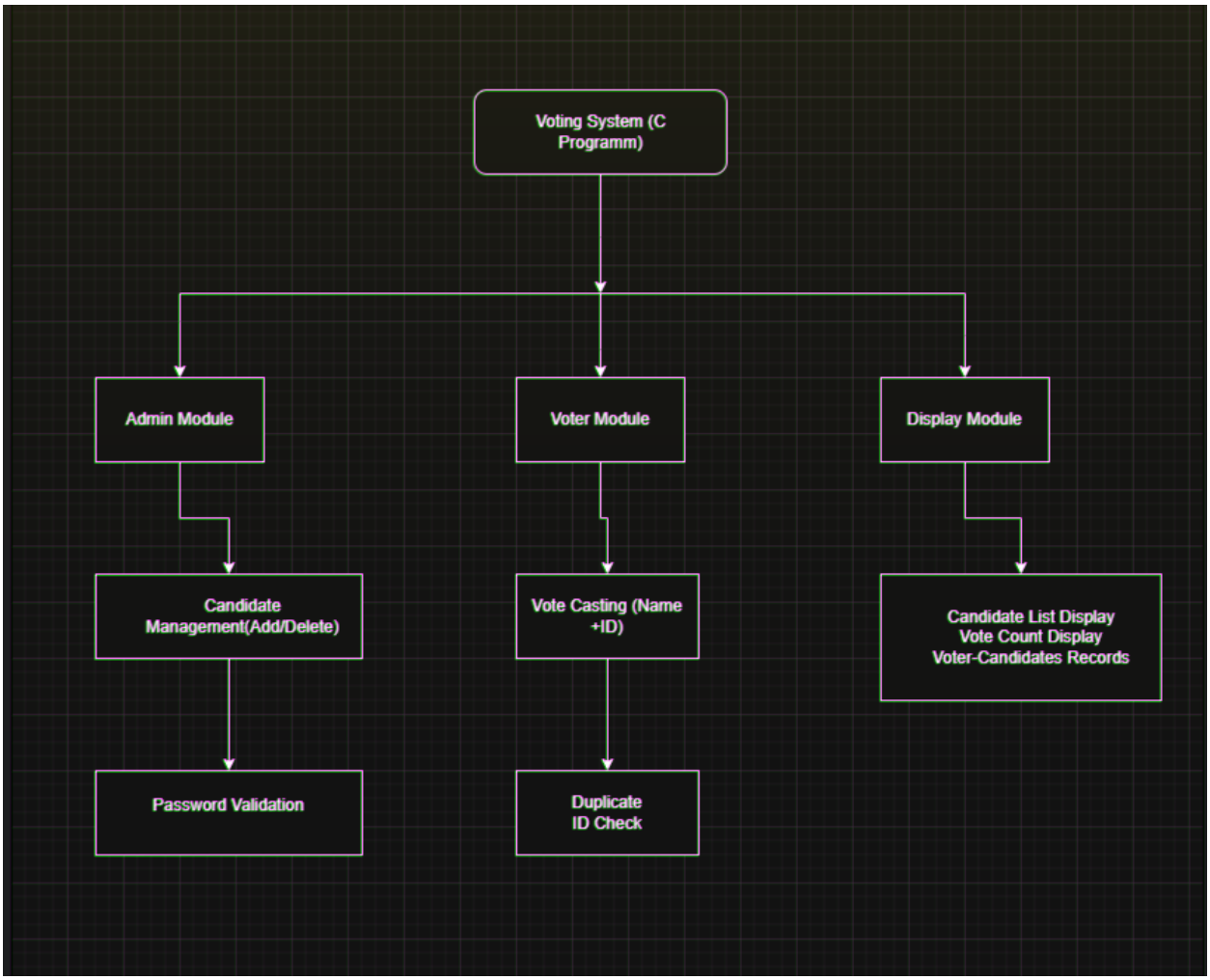
OBJECTIVES

- To develop a straightforward and effective console-based voting application using the C programming language.
- To enable administrators to add, delete, and remove candidates through password-protected security
- To allow users to vote with their name and their personal 13-digit code while restricting repeat voters from voting again.
- To remain transparent by providing a list of candidates, results, and detailed logs of voter activity.
- To show the functionality of arrays, strings, loops, and user-defined functions to solve real-world problems

- To design an intuitive UI to simplify the voting experience for both the administrator and the general user.

SYSTEM DESIGN

System Overview



ALGORITHM

1. Start the program
2. Initialize

   - Create arrays for candidate names, voter names, IDs, and vote choices.
   - Set `candidateCount = 0`, `total_user = 0`, and initialize votes to zero.

3. Display Main Menu

    - Add Candidates (Admin Only)
    - Delete Candidates (Admin Only)
    - Vote for Candidate
    - View list of Candidates
    - Show Results
    - Show who voted for whom
    - Exit

4. Input User Choice
5. Execute According to User Choice

    i. If choice =1: Add Candidates (Admin)

        - Ask for admin Password
        - If incorrect → Show error and return to Menu
        - If correct → Allow adding Candidates names
        - Validate Candidate names (No Digits)
        - Store name in array and increment `candidateCount`

    ii. If choice = 2: Delete Candidate (Admin)

        - Ask for Admin Password
        - If incorrect → Return to menu
        - If correct → Display list of candidates
        - Input Candidate number to delete
        - Shift array elements to delete that candidate
        - Decrement `candidateCount by 1`

    iii. If Choice = 3: Voting

        - Ask voter's name
        - Ask voters for their 13-digit ID
        - Validate ID format.
        - Check if ID already exists in voter records
        - If yes → reject vote and return to menu.
        - Display list of candidates.
        - Ask for vote selection.
        - Validate choice.
        - Record votes for candidate and store: Voter name, Voter ID, Voter choice.
        - Increment `total_user`.

    iv. If Choice = 4: View Candidates

- Display all candidate names with numbering.

v. If choice = 5: Show Results

- Display each candidate and vote count.
- Identify and print the candidate with maximum votes.

vi. If choice = 6: Show Who Voted for Whom

- For each user, print: voterName → candidateName voted for

vii. If choice = 7: Exit

- Terminate program.

6. Loop back to Step 3 until user selects Exit
7. Exit

INPUT & OUTPUT

Input

- Admin Password
- Candidate Name
- Voter Name
- 13 Digit Voter ID
- Menu Choices
- Vote Choice
- Candidate Number for Deletion

Output

- Succes/Error Messages
- Candidate List
- Vote Confirmation
- Voting Results
- Voter summary
- Deletion Output
- Exit Message

IMPLEMENTATION

Language: C

Compiler/IDE: GCC(MinGW)/VS Code

Key Features

- Admin Authentication
- Candidate Management
- Unique Voter Identification
- User Friendly Voting Process
- Real Time Vote counting
- Detailed Vote Tracking
- Candidate Display & Result View
- Modular Function-Based Design
- Input Validation
- Clear console interface

Code Snippets

View Candidate Function

```c
void viewCandidates(char candidateNames[20][30], int candidateCount) {
    if (candidateCount == 0) {
        printf("No candidates available.\n");
        return;
    }
    printf("\n----- List of Candidates -----\n");
    for (int i = 0; i < candidateCount; i++) {
        printf("%d. %s\n", i + 1, candidateNames[i]);
    }
    printf("------------------------------\n");
}
```

Who Voted for Whom Function

```c
void showVotes(char voterNames[100][20], int voteChoice[100], char candidateNames[20][30], int total_user) {
    if (total_user == 0) {
        printf("No votes cast yet.\n");
        return;
    }

    printf("\n----- Who Voted For Whom -----\n");
    for (int i = 0; i < total_user; i++) {
        printf("%s voted for %s\n", voterNames[i], candidateNames[voteChoice[i]]);
    }
    printf("------------------------------\n");
}
```

All Functions

```c
int addcandidate(char candidateNames[20][30], int candidateCount);
void viewCandidates(char candidateNames[20][30], int candidateCount);
void showResults(char candidateNames[20][30], int votes[20], int candidateCount);
void deleteCandidate(char candidateNames[20][30], int *candidateCount, int votes[20]);
int isValidName(char *name);
void showVotes(char voterNames[100][20], int voteChoice[100], char candidateNames[20][30], int total_user);
//FILING FUNCTIONS
void saveCandidateToFile(char *candidate);
void saveVoterToFile(char *name, long long id);
void saveVoteRecord(char *voterName, char *candidateName);
void rewriteCandidatesFile(char candidateNames[20][30], int candidateCount);
int loadCandidatesFromFile(char candidateNames[20][30]);
int loadVotersFromFile(long long idList[100]);
```

TESTING & RESULT

Test Case 1: Add Candidate (Valid Input)

Input Password: `admin123`, Candidate Name: `John Khan`

Expected Output “Candidate Added: John Khan”

Actual Output As expected,

Status Pass

Test Case 2: Add Candidate (Invalid Password)

Input Password: wrong pass

Expected Output “Wrong password! Access denied.”

Actual Output As expected,

Status Pass

Test Case 3: Add Candidate (Invalid Name)

Input Name: John123

Expected Output “Invalid name! Numbers are not allowed.”

Actual Output As expected,

Status Pass

Test Case 4: Delete Candidate (Valid Delete)

Input	Password: admin123, Delete Candidate 2
Expected Output	“Candidate ‘X’ deleted successfully.”
Actual Output	As expected,
Status	Pass

Test Case 5: Delete Candidate (Invalid Number)

Input	Candidate number: 10 (when only 3 exist)
Expected Output	“Invalid candidate number.”
Actual Output	As expected,
Status	Pass

Test Case 6: Voting (Valid Vote)

Input	Voter Name: Ali, ID: 4210187654321, Vote: 1
Expected Output	“Vote recorded successfully for <candidate>!”
Actual Output	As expected,
Status	Pass

Test Case 7: Voting (Invalid ID - not 13 digits)

Input	ID: 123456
Expected Output	“Invalid ID.”
Actual Output	As expected,
Status	Pass

Test Case 8: Voting (Duplicate ID)

Input	ID already used in previous vote
Expected Output	“This ID has already voted!”
Actual Output	As expected,
Status	Pass

Test Case 9: Voting (Invalid Candidate Choice)

Input	Vote: 5 (when only 3 candidates exist)
Expected Output	“Invalid choice! Vote not recorded.”
Actual Output	As expected,
Status	Pass

Test Case 10: View Candidates

Input	Choose “View Candidates”
Expected Output	List of all candidates displayed
Actual Output	As expected,
Status	Pass

Test Case 11: Show Results

Input	Choose “Show Results”
Expected Output	Each candidate shown with vote count & winner identified
Actual Output	As expected,
Status	Pass

CONCLUSION, LIMITATION & REFRENCENCES

Conclusion

In summary, the designed voting system effectively illustrates the usefulness of basic concepts of programming in creating a practical and functional program. The system uses modularity, logical structure, and efficient data management to provide users with functionality for candidate registration, voting, and results. The program includes security with admin authentication, and its data validation also incorporates input to protect voter data integrity. The system is basic in nature, yet fulfills the goals securely, and can easily transition to more advanced electronic voting systems with enhanced data storage, interfaces, and overall security features.

Limitation

- Limited to Fixed Array Sizes
- Console-Based Interface Only
- No Advanced Security Features

Future Enhancements

- Adding File handling/Data base to store data more efficenty.
- Graphical User Interface (GUI)
- Scalability and Dynamic Data Structures