Project report

ELECTRONIC VOTING SYSTEM

Empowering Secure, Transparent, and Accessible Elections.

TEAM MEMBERS:

Ghulam Murtaza *K240720*
Mehrwan *k240864*
Sanaullah *k240954*

Submitted to:

Ms.Khadija tul Kubra

NATIONAL UNIVERSITY of Computer & Emerging Sciences

CL1002-Programming Fundamentals

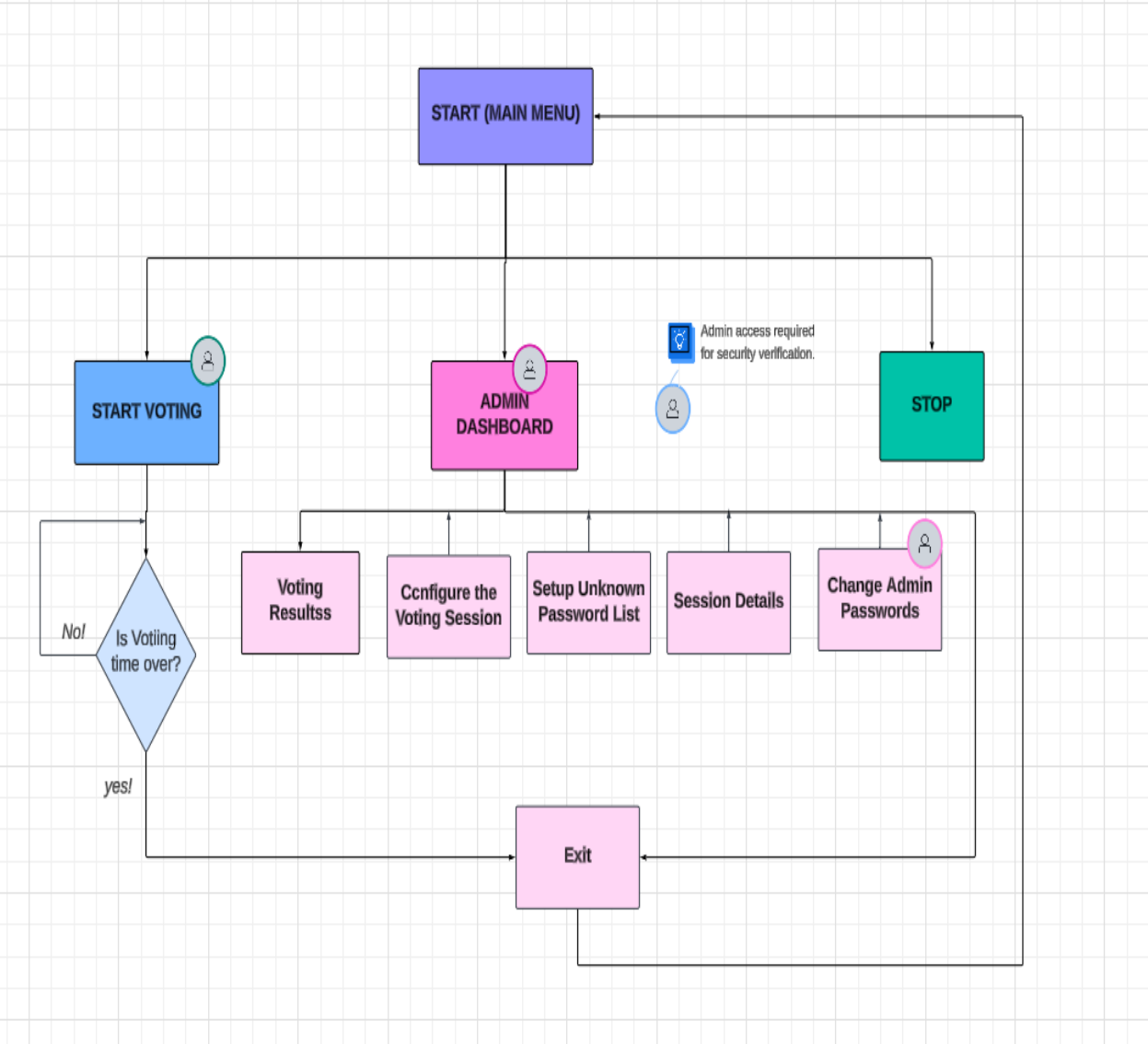
Introduction

“Our team project is an Electronic Voting System in C. This system aims to make the voting process more efficient and secure by digitizing it, with a user-friendly interface for both administrators and voters. The admin can easily manage data, set up the voting session, and maintain security by saving passwords in an encrypted format. We use XOR and other operations on the encryption key to ensure that stored data remains protected and difficult to crack.

Voters can securely log in, cast their votes, and have their selections saved in organized files, which helps. with accurate result management. We chose this project because traditional voting methods often encounter issues like manual errors and delays. By automating and securing the process, this electronic voting system can help address these challenges, making it a timely and relevant solution today. “

Design & Implementation:

Flow Chart of Main Menu Which helps to Understand the flow of Process Overall.



Pseudo Code for Main Menu (Big Picture of Electronic Voting System):

START Program

DISPLAY Main Menu
  OPTION 1: "Start Voting"
  OPTION 2: "Admin Dashboard"
  OPTION 3: "Stop"

IF user selects "Start Voting" THEN
  PROMPT user to enter ID and Password
  IF authentication fails THEN
    DISPLAY "Invalid credentials, please try again."
  ELSE
    WHILE voting session is active
      DISPLAY candidate options
      PROMPT user to select a candidate
      RECORD vote
      CHECK if voting time is over
    END WHILE
    DISPLAY "Voting session has ended."
  END IF

ELSE IF user selects "Admin Dashboard" THEN
  PROMPT for Admin Password
  IF Admin Password is correct THEN
    DISPLAY Admin Options:
      OPTION A: "View Voting Results"
      OPTION B: "Configure Voting Session"
      OPTION C: "Setup Unknown Password List"
      OPTION D: "View Session Details"
      OPTION E: "Change Admin Password"
      OPTION F: "Exit"

    CASE (Admin Option Selection)
      "View Voting Results" ->
        DISPLAY voting results
      "Configure Voting Session" ->
        PROMPT to set up session details
        (time, candidates, etc.)
      "Setup Unknown Password List" ->
        ALLOW admin to configure or add passwords
      "View Session Details" ->
        DISPLAY session timing and active status
      "Change Admin Password" ->
        PROMPT to update admin password
      "Exit" -> RETURN to Main Menu
    END CASE
  ELSE
    DISPLAY "Access denied: Admin password incorrect."
  END IF

ELSE IF user selects "Stop" THEN
  EXIT Program

```c
void display_main_menu(){
    int menu_option;
    system("cls");
    printf("\n\n\n\n\n");
printf("\t\t\t\t\t   EEEEEEEEEE   VVV       VVV   SSSSSSSS   \n");
printf("\t\t\t\t\t   EEEEEEEEEE   VVV       VVV  SSSSSSSSSS   \n");
printf("\t\t\t\t\t   EEE           VVV     VVV   SSS\n");
printf("\t\t\t\t\t   EEE           VVV     VVV    SSS\n");
printf("\t\t\t\t\t   EEEEEEEE       VVV   VVV      SSSSSSS\n");
printf("\t\t\t\t\t   EEEEEEEE       VVV   VVV       SSSSSSS\n");
printf("\t\t\t\t\t   EEE             VVV VVV            SSS\n");
printf("\t\t\t\t\t   EEE             VVV VVV            SSS\n");
printf("\t\t\t\t\t   EEEEEEEEEE       VVVVV        SSSSSSSSS\n");
printf("\t\t\t\t\t   EEEEEEEEEE        VVV         SSSSSSSSS\n");


    printf("\n\n\t\t\t\t\t   Choose an option from the menu below:\n");
    printf("\t\t\t\t\t   ====================================\n");
    printf("\t\t\t\t\t   1. Start Voting\n");
    printf("\t\t\t\t\t   2. Admin Dashboard\n");
    printf("\t\t\t\t\t   3. Exit\n");
    printf("\t\t\t\t\t   ====================================\n");

    printf("\t\t\t\t\t   Enter Menu Option i.e 1,2,3 : ");
    scanf("%d",&menu_option);
    if(menu_option==3){
        exit(0);
    }
```

```c
    printf("\t\t\t\t\t   Enter Menu Option i.e 1,2,3 : ");
    scanf("%d",&menu_option);
    if(menu_option==3){
        exit(0);
    }
    else if(menu_option >3 || menu_option<0){
        printf("\t\t\t\t\t   Invalid Input Menu\n");
        printf("\t\t\t\t\t   Enter Menu Option i.e 1/2/3 : ");
        scanf("%d",&menu_option);
    }
    if(menu_option==1)
    display_voting_menu();
    if(menu_option==3){

        return;
    }
    else if(menu_option >3 || menu_option<0){
        printf("\t\t\t\t\t   Invalid Input Menu\n");
        printf("\t\t\t\t\t   Enter Menu Option i.e 1/2/3 : ");
        scanf("%d",&menu_option);
    }
    if(menu_option >3 || menu_option<0){
        printf("\t\t\t\t\t   Invalid Input Menu, limit Exceed");
    }
    if(menu_option == 2){
    key=1;
    menu_call();
    }
```

END IF
END Program

The Electronic Voting System is designed to make voting sessions secure and easy for roles like "Class Representative", "President of University Societies" …Etc. The admin sets up the session with details like the election role, candidates, eligible voters, and voting time. Voters log in, choose their candidate, and cast their vote, which is safely stored in the system. The admin's access is secured with an encrypted password to protect session settings and voter data. The system keeps track of the session time, showing voters, how much time is left. This setup aims to make voting faster, reduce errors, and ensure a fair process for everyone involved.

Pseudo Code for Admin Dashboard:

Function display_admin_menu:
Clear screen
Display ASCII art for "Admin Dashboard"
Display menu title based on menu_option value:
If menu_option is 1:
Show message: "Before Starting the Voting, Please Login to Verify"
Else:
Show message: "Setup the Dashboard belo
Display separator line

Function menu_call:
Call display_admin_menu to show the dashboard
If key is set to 1:
Call admin_login to authenticate the admin user

Declare variable 'selection' to store menu choice
Display admin menu options:
1. Voting Results
2. Configure Voting Session
3. Setup Unknown Password List
4. Session Details
5. Change Admin Password
6. Exit

Prompt the user to enter a menu option and store it in 'selection'

Call display_admin_menu to refresh the screen

Based on the 'selection' value, perform the corresponding action:
If selection is 1:
Call main_result to display voting results

If selection is 4:
Call current_session to display session details

If selection is 6:
Call display_main_menu to return to the main menu
Set key to 1 to require login again

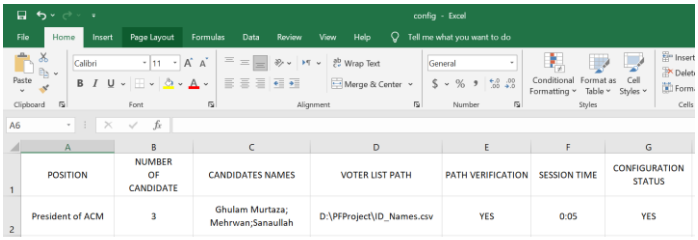
If selection is 5:
Call change_password_4 to change the admin password

If selection is 2:
Call config_main to configure the voting session

If selection is 3:
Declare 'currentPath' to store the directory path
Initialize the random number generator for password generation
Call driveSelection with 'currentPath' to set up unknown password list.

The core of this system relies on different files that store and organize important data related to voter IDs, passwords, candidates, and voting results. Before diving into the specifics of the Admin Dashboard and its options, it's crucial to understand how these files storage pattren, as they form the backbone of the program's functionality.

- Config.csv
  This file holds the configuration settings for the voting session (e.g., session time candidates).



	POSITION	NUMBER OF CANDIDATE	CANDIDATES NAMES	VOTER LIST PATH	PATH VERIFICATION	SESSION TIME	CONFIGURATION STATUS
2	President of ACM	3	Ghulam Murtaza; Mehrwan;Sanaullah	D:\PFProject\ID_Names.csv	YES	0:05	YES

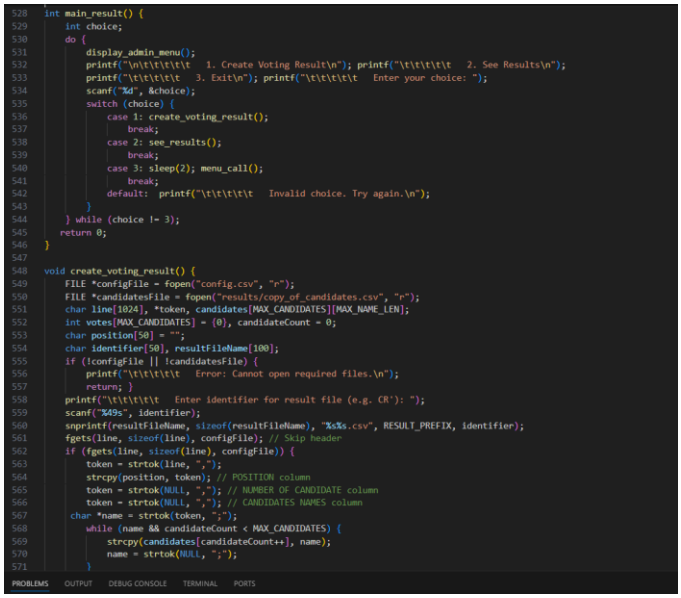
- Copy_of_candidates.csv
  After the Setup Unknown password list. This file lists all the candidates running in the election, which will be displayed to voters during the election also in this file the two new columns will be created automatically where the candidate's selection and status will be show.

	ID	NAME	PASSWORDS	Vote Status	CANDIDATE SELECTED
2	24K-0720	Rehman	z&vOkvBU	NOT VOTED	Not Selected
3	24K-0864	Meharwar	fDyooS%L	NOT VOTED	Not Selected
4	24K-0512	Aly Muhar	g2sgf8*o	NOT VOTED	Not Selected
5	24K-0643	Syed Abra	2kW5*dWx	NOT VOTED	Not Selected
6	24K-0606	ASHAR AD	X%X%NzYc	NOT VOTED	Not Selected
7	24K-1030	Muhamm	*nq8$fnk	NOT VOTED	Not Selected
8	24K-1027	Noman Al	sd^1rPGW	NOT VOTED	Not Selected
9	24K-0911	Faraz	uVUoARaV	NOT VOTED	Not Selected
10	24K-0853	Hafiz Muh	Y#X1x*JW	NOT VOTED	Not Selected
11	24K-0694	Hashir Ali	x$d4glm^	NOT VOTED	Not Selected
12	24K-0994	Izaan Khar	#9wO7A^f	NOT VOTED	Not Selected

- Result CSV Files
  These are the final result files generated after the voting session is complete, like `resultPresident.csv'.

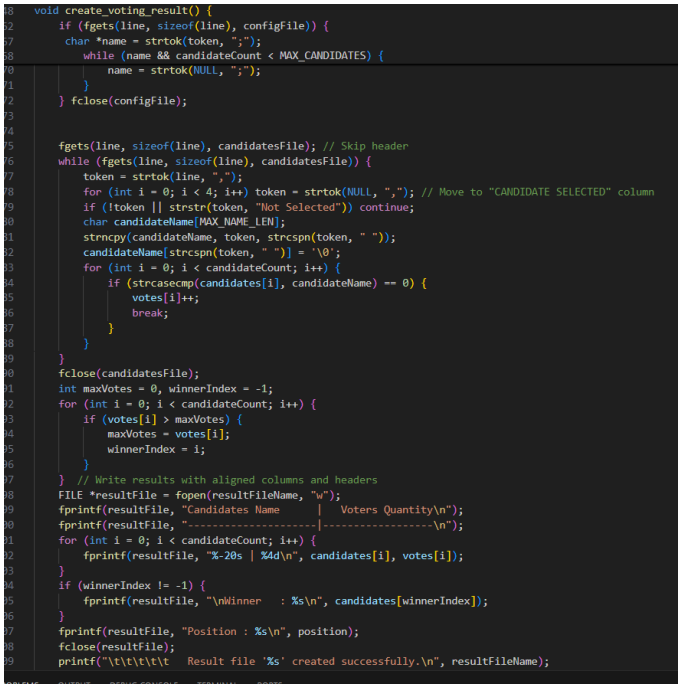
	A	B	C	D
1	Candidates Name		Voters Quantity	
2	----------		----------	
3	Ghulam Murtaza		4	
4	Mehrwan		0	
5	Sanaullah		1	
6				
7	Winner	:	Ghulam Murtaza	
8	Position	:	President of ACM	

Now you got a knowledge about how files are store ,Now you can easily understand how the functions work in admin Dashboard lets dive in to Result functi

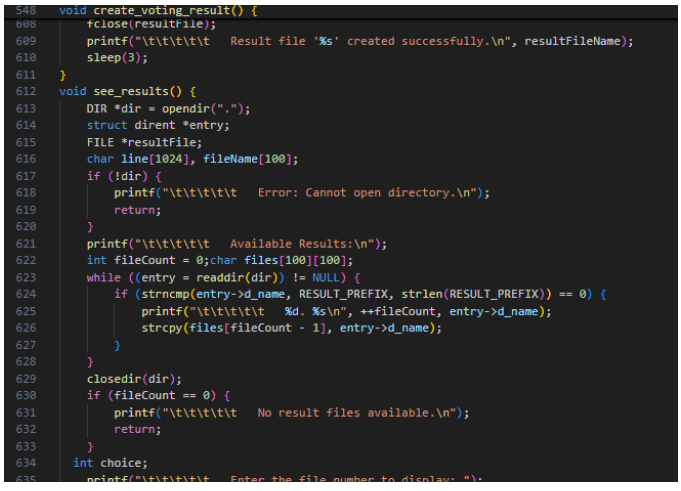


Pseudo Code for Voting Results:

main_result:
- Repeat until choice is 3:
  - Display admin menu options:
    1. Create Voting Result
    2. See Results
    3. Exit
  - Prompt user to enter a choice
  - If choice is 1:
    - Call create_voting_result
  - Else if choice is 2:
    - Call see_results
  - Else if choice is 3:
    - Exit and return to main menu
  - Otherwise:
    - Show message: "Invalid choice, try again."



create_voting_result:
- Open config file and candidates file
- If files cannot be opened:
  - Show error message and exit function
- Prompt admin for a file identifier to name the resul
- Read position and candidate names from the conf
- Initialize candidate names and votes count
- For each line in candidates file:
  - Get the name of the selected candidate
  - If candidate is valid, increase their vote count
- Determine the winner with the most votes
- Open a new result file and write candidate names and vote counts
- Write the winner's name and position to the file
- Show message that the result file was created suc
- Close files



- see_results:
  - Open the current directory to search for result files
  - If directory cannot be opened:
    - Show error message and exit function
  - Display a list of result files starting with a specific p
  - If no result files are found:
    - Show message: "No result files available."
    - Exit function
  - Prompt user to select a file to view
  - If choice is invalid:
    - Show message: "Invalid choice."
    - Exit function

Open the selected file and display its contents (candidates and vote counts)
Show winner and position details
Close file and wait for user to press any key to exit

```
int choice;
  printf("\t\t\t\t\t   Enter the file number to display: ");
  scanf("%d", &choice);
  sleep(2);
      display_admin_menu();
  if (choice < 1 || choice > fileCount) {
      printf("\t\t\t\t\t   Invalid choice.\n");
      return;
  }
  resultFile = fopen(files[choice - 1], "r");
  if (resultFile) {
      printf("\n\t\t\t\t\t   File: %s\n", files[choice - 1]);
      printf("\t\t\t\t\t   ");printf("Candidates Name      |   Voters Quantity\n");
      fgets(line, sizeof(line), resultFile); // Skip header
      while (fgets(line, sizeof(line), resultFile)) {
      printf("\t\t\t\t\t   ");    printf("%s", line);
      }
      fclose(resultFile);
  }
  printf("\n\t\t\t\t\t   Enter any key to Exit.");
  sleep(4);
  fflush(stdin);
  getchar();
}
```

Pseudo Code for Configure Voting Session:

config_main:

- **Repeat loop until configuration is saved or user exits**:
  - If 'k' is 1:
    - Show admin menu and call config_main again.
    - Get position of candidates from the user using get_position.
    - Get the number of candidates using get_number_of_candidates.
    - For each candidate, get their name using get_candidate_names.
    - Get the path for the voter list file using get_voter_list_path.
    - **Verify if the voter list path is valid**:
      - If path is verified:
        - Get session time (hours and minutes) using get_session_time.
        - **Check if configuration is valid** using validate_configuration.
        - Save all the information to a CSV file using save_configuration_to_csv.
        - Show confirmation message: "Configuration saved successfully!"
        - Exit loop and return to the main menu.
        - Otherwise:
          - Show error message: "Voter list path verification failed."
          - Ask user if they want to retry (Y/N).
            - If user chooses 'N':
              - Return to the main menu.

```
int config_main(){
    // this are the Variables to store input data -------> Configue
    char position[50];int num_candidates;
    char candidate_names[MAX_CANDIDATES][maxname_leng];
    char voter_list_path[256];int hours, minutes;
    int is_verified = 0; int is_configured = 0;
    int k = 0;char per;
    for(k=0;k!=2;){
    if(k==1)
    {
    display_admin_menu();
    config_main();
    }// Get inputs--->Separatelyyy
    get_position(position);
    get_number_of_candidates(&num_candidates);
    get_candidate_names(candidate_names, num_candidates);
    get_voter_list_path(voter_list_path);
    // Verify the path USIng this Funciton
    is_verified = verify_path(voter_list_path);
    if (is_verified) {
        get_session_time(&hours, &minutes);
            // Validate --? configuration
        is_configured = validate_configuration(num_candidates, candidate_names);
    // Save configuration to CSV file
     save_configuration_to_csv(position, num_candidates, candidate_names,
        voter_list_path, is_verified, hours, minutes, is_configured);
        printf("\n\t\t\t\t  Configuration saved successfully!\n "); menu_call(); //Again MENUUUU CA
        getchar(); k=2;
    }
else {
        printf("\n\t\t\t\t\t    Error: Voter list path verification failed.\n");
        printf("\t\t\t\t\t    Do you try again config Y/N : ");
        scanf("%c",&per);
        k=1;
        if(per=='N'|| per=='n')
        menu_call();
    }
    }
    return 0;
}
```

get_position:

- **Prompt user to enter the position for the candidates**.
- Remove any extra newline from input.

```
void get_position(char *position) {
    printf("\t\t\t\t\t         Position for candidates are running: ");
    getchar();
    fgets(position, 50, stdin);
    position[strcspn(position, "\n")] = '\0'; // Remove newline
}
```

get_number_of_candidates:

- **Prompt user to enter the number of candidates**.
- If input is invalid (e.g., number exceeds limit or is negative):
  - Show "Invalid Input" message.
  - Repeat prompt.

```
void get_number_of_candidates(int *num_candidates) {
    printf("\t\t\t\t\t         Enter the number of candidates: ");
    scanf("%d", num_candidates);int random=*num_candidates;
    if(random>MAX_CANDIDATES || random<0)
    {printf("\n\t\t\t\t\t           Invalid Input\n");
    get_number_of_candidates(num_candidates);
    } getchar();
}
```

get_candidate_names:

- For each candidate:
  Prompt user to enter the candidate's name.
- Remove any extra newline from input.

```
void get_candidate_names(char candidate_names[][maxname_leng], int num_candidates) {
    for (int i = 0; i < num_candidates; i++) {
        printf("\t\t\t\t\t         Enter name of candidate %d: ", i + 1);
        fgets(candidate_names[i], maxname_leng, stdin);
        candidate_names[i][strcspn(candidate_names[i], "\n")] = '\0';
    }
}
```

get_voter_list_path:

- **Prompt user to enter the file path for the voter list**.
- Remove any extra newline from input.

```
void get_voter_list_path(char *voter_list_path) {
    printf("\n\t\t\t\t\t         Enter path of voter list CSV file i.e D:\\3uToolsV3\\list.csv : ");
    fgets(voter_list_path, 256, stdin);
    voter_list_path[strcspn(voter_list_path, "\n")] = '\0'; // Remove newwwwline
}
```

verify_path:

- Open the file at the given path.
  - If the file cannot be opened:
    - Return 0 (invalid path).
  - Read the first line of the file.
  - **Count the columns by checking the number of comn**
  - If there are exactly 3 columns:
    - Return 1 (valid path).
  - Otherwise:
    - Return 0 (invalid path).

```
int verify_path(const char *path) {
    FILE *file = fopen(path, "r");
    if (file == NULL) { return 0;
    }char line[256];
    fgets(line, sizeof(line), file);
    fclose(file); int columns = 1; // Start with 1 column
    for (int i =0; line[i]!='\0'; i++) {  if (line[i] == ',') {
            columns++;} }
    return columns == 3 ? 1 : 0;
}
```

get_session_time:

- **Prompt user to enter session duration** (hours and minutes).
- Show a message:
  "Successfully Done Configuration".

```
void get_session_time(int *hours, int *minutes) {
    printf("\t\t\t\t\t         Enter session duration (hours minutes): ");scanf("%d %d", hours, minutes);
    sleep(1);printf("\t\t\t\t\t         Successfully Done Configuration ");
    sleep(1);
}
```

validate_configuration:

- If the number of candidates is not positive
  - Return 0 (invalid configuration).
- For each candidate:
  - If the candidate's name is empty:
    --Return 0 (invalid configuration).
- Return 1 if all conditions are met (valid configuration).

```
int validate_configuration(int num_candidates, char candidate_names[][maxname_leng]) {
    if (num_candidates <= 0) return 0;
    for (int i= 0; i <num_candidates; i++) {
        if (strlen(candidate_names[i]) == 0) return 0;
    }
    return 1;
}
```

save_configuration_to_csv:

- Open or create a new CSV file to save the configuration.
  - If the file cannot be opened:
    - Show error message and return.
  - **Write position, number of candidates, candidate names, voter list path, verification status,session time, and configuration status** to the file.
  - Close the file.

```c
void save_configuration_to_csv(const char *position, int num_candidates, char candidate_names[][maxname_leng],
                               const char *voter_list_path, int is_verified, int hours, int minutes, int is_configured) {
    FILE *file = fopen(FILE_PATH, "w");
    if (file == NULL) {//Again checking for NULl
        printf("\t\t\t\t\t        Error: Could not open file to save configuration.\n");
        return;
    }
    fprintf(file, "POSITION,NUMBER OF CANDIDATE,CANDIDATES NAMES,VOTER LIST PATH,PATH VERIFICATION,SESSION TIME,CONFIGURATION STATUS\n");
    fprintf(file, "%s,%d,", position, num_candidates);
      for (int i = 0; i < num_candidates; i++) {
        fprintf(file, "%s", candidate_names[i]);
        if (i < num_candidates - 1) {
            fprintf(file, ";");
        }
    }
    fprintf(file, ",%s,%s,%02d:%02d,%s\n",
            voter_list_path,
        is_verified ? "YES" : "NO",
            hours, minutes,
        is_configured ? "YES" : "NO");
    fclose(file);
}
```

Pseudo Code for Setup Unknown Password List:

```c
void listDrives() {
    DWORD drives = GetLogicalDrives();
    printf("\t\t\t\t\t    Available Drives:\n");
    for (char i = 'A'; i <= 'Z'; i++) {
        if (drives & (1 << (i - 'A'))) {
            printf("\t\t\t\t\t    Drive: %c:\n", i); }
    }
}
```

listDrives:

- Get a list of all logical drives.
- Loop through all drives (from A to Z).
- For each drive, check if it's available:
  - If it is, print the drive letter (e.g., C:).

---

listDirectories:

- Print the directories in the specified path.
- Search for directories using the FindFirstFile function.
- If directories are found:
  - Loop through the directories and print their names.
  - Skip. and .. directories.
- Wait for the user to note down the directory names.
- After a short pause, finish the directory listing.

```c
void listDirectories(const char *path) {
    printf("\n\t\t\t\t\t  Directories in %s:\n", path);
    WIN32_FIND_DATA findFileData;
    char searchPath[max_path];
    snprintf(searchPath, max_path, "%s\\*", path);
    HANDLE hFind = FindFirstFile(searchPath, &findFileData);
    if (hFind == INVALID_HANDLE_VALUE) {
        printf("\t\t\t\t\t  Error finding directories.\n");
        sleep(2); return;
    }
    do {
        if (findFileData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) {
            if (strcmp(findFileData.cFileName, ".") != 0 && strcmp(findFileData.cFileName, "..") != 0) {
                printf("\t\t\t\t\t  Directory: %s\n", findFileData.cFileName);
            }
        }
    } while (FindNextFile(hFind, &findFileData) != 0);sleep(2);
    printf("\t\t\t\t\t  Waiting for 5 sec (note the name of directory) ");sleep(5);
    FindClose(hFind);
}
```

---

listCSVFiles:

- Print all CSV files in the specified path.
- Search for .csv files using the FindFirstFile function.
- If CSV files are found:
  - Loop through the files and print their names.
  - Let the user select a file by entering a number.
- Validate the selection:
  - If the user selects an invalid file, print an error message.
- Add a password column to the selected CSV file using addPasswordColumnToCSV.

```c
void listCSVFiles(const char *path) {
printf("\n\t\t\t\t\t      CSV Files in %s:\n\t\t\t\t\t      ", path);
    WIN32_FIND_DATA findFileData;
    char searchPath[max_path];
    snprintf(searchPath, max_path, "%s\\*.csv", path);   //print path krny ky liye
    HANDLE hFind = FindFirstFile(searchPath, &findFileData);
    if (hFind == INVALID_HANDLE_VALUE) {
        printf("\t\t\t\t\t  No CSV files found in the current directory.\n");
        sleep(2); // screen stop for 2sec here
        return;
    }
    int count = 0;
    do {
        printf("\t\t\t\t\t      %d. %s\n", ++count, findFileData.cFileName);
    } while (FindNextFile(hFind, &findFileData) != 0);
    FindClose(hFind);
    if (count == 0) {
    printf("\t\t\t\t\tNo CSV files found in the current directory.\n");
    sleep(2); return;
    }
    int fileChoice;
    printf("\t\t\t\t\t      Enter the number of the CSV file to modify: ");
    scanf("%d", &fileChoice);
    sleep(2);
    hFind = FindFirstFile(searchPath, &findFileData);
    for (int i = 1; i < fileChoice && FindNextFile(hFind, &findFileData); i++);
    if (fileChoice <= 0 || findFileData.cFileName[0] == '\0') {
        printf("\t\t\t\t\t Invalid selection.\n");
        sleep(2);
        FindClose(hFind); //closing the file
        return;
    }
    char selectedFile[max_path]; //selected file path
    snprintf(selectedFile, max_path, "%s\\%s", path, findFileData.cFileName);
    if (addPasswordColumnToCSV(selectedFile) == 0) {
printf("\t\t\t\t\tPassword column added successfully to %s.\n", selectedFile);
        sleep(2);
    }
```

---

generatePassword:

- Generate a random 8-character password using a set of characters

```c
void generatePassword(char *password) {
const char *charset = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789!@#$%^&*";
    for (int i =0; i< password_length; i++) {
        password[i] = charset[rand() % strlen(charset)];
    }
    password[password_length] = '\0';/
}
```

(uppercase, lowercase, digits, and special characters).
- Assign the password to the provided string.

---

addPasswordColumnToCSV:

- Open the CSV file for reading.
  - If the file cannot be opened, print an error and return.
- Create a temporary file to write the updated CSV data.
- Read the CSV file line by line:
  - For the first row (header), just copy it to the temporary file.
  - For subsequent rows:
    - Split each line into columns (ID, Name, etc.).
    - Add a randomly generated password in the third column.
- Write the updated line to the temporary file.
- After processing all lines:
  - Replace the original CSV file with the updated one.

```
int addPasswordColumnToCSV(const char *filePath) {
    FILE *file = fopen(filePath, "r"); //OPen the File as r mode
if (file == NULL) { //if file is Null
    printf("\t\t\t\t\t Error opening the CSV file at %s.\n", filePath);
        return -1;
    }   //checking
    FILE *tempFile = fopen("temp.csv", "w");
    if (tempFile == NULL) {
        printf("Error creating temporary file.\n");
       fclose(file);
    return -1;
    }
char line[linemaxlength];char password[password_length + 1];
    int isFirstRow = 1;// Process With tHE CSV line by line
    while (fgets(line, sizeof(line), file)) { // If it's the header row Just copy it
        if (isFirstRow) {
        fputs(line, tempFile);  isFirstRow = 0;
        } else { // the Generation pass ANDD append it to the third column
            generatePassword(password);
            char *token = strtok(line, ",");
            char newLine[linemaxlength] = "";
            int colCount = 0;// Itterate tHrough each column Preserving the first two columns (ID and Name)
            while (token != NULL) {  // Keep the ID and Name columns as they are
                if (colCount == 2) {  // This ISSS the third column (Passwords) idhar hy password
                    strcat(newLine, ",");
                    strcat(newLine, password);  // Add password to the third column ---> through ConaAtnate
                    break;
                }               if (colCount > 0) {  // Add commas for the other columns-----> throught string concatinate Function
                    strcat(newLine, ",");
                }
                strcat(newLine, token);
                colCount++;
                token = strtok(NULL, ",");
            }
            // ADD append the new ------------>  temporary file
            strcat(newLine, "\n");
            fputs(newLine, tempFile);
        }
    }
    fclose(file);
    fclose(tempFile);
    // Replace the original CSV with the updated one
    if (remove(filePath) != 0 || rename("temp.csv", filePath) != 0) {
        printf("\t\t\t\t\t  Error updating the CSV file.\n");
        return -1;
    }
    return 0;
}
```

isValidDrive:

- Check if the selected drive letter corresponds to a valid drive.
  - If the drive exists and is a directory, return true; otherwise, return false.

```
// --------> Validation of the Drive ----Function to check if the selected drive is valid
int isValidDrive(char driveLetter) {
    char drivePath[4];//
    snprintf(drivePath, sizeof(drivePath), "%c:\\", driveLetter);

DWORD attributes = GetFileAttributes(drivePath);
    return (attributes != INVALID_FILE_ATTRIBUTES && (attributes & FILE_ATTRIBUTE_DIRECTORY)); //return the directory

}
```

---

driveSelection:

- Ask the user to select a drive by entering a letter (e.g., C).
- Validate the selected drive using isValidDrive.
  - If valid, set the current path to the selected drive.
  - If invalid, prompt the user to try again.
- Call showMenu to display the menu for further actions.

```
void driveSelection(char *currentPath) {
    char driveLetter;
    while (1) {
        printf("\t\t\t\t\t   Select a Drive:\n");
    listDrives(); //---> everytime this function is use for lisitng
        printf("\t\t\t\t\t    Enter drive letter (e.g., C): ");
        scanf(" %c", &driveLetter);

    // Checking ----------> for if the entered drive is valid
        if (isValidDrive(driveLetter)) {
            snprintf(currentPath, max_path, "%c:\\", driveLetter);
            break;  // Exit loop ----> if tHE drive is valid
        } else {
            printf("\t\t\t\t\t Invalid drive letter, Please try again.\n");
        }
    }
```

showMenu:

- Display the menu options:
  - List directories
  - List CSV files
  - Enter a directory
  - Go back to the parent directory
  - Change the drive
  - Exit the program
- Prompt the user to choose an option.
- Based on the user's choice:
  - List directories using listDirectories.
  - List CSV files using listCSVFiles.
  - Change directory using changeDirectory.
  - Go back to the parent directory using goBackDirectory.
  - Change the drive using driveSelection.
- After completing the action, show the menu again recursively.

```
void showMenu(char *currentPath) {
    int choice;

    display_admin_menu();
    printf("\n\t\t\t\t\t\t    Current Directory: %s\n", currentPath);
    printf("\t\t\t\t\t\t    Options:\n");
    printf("\t\t\t\t\t\t      1. List Directories\n");
    printf("\t\t\t\t\t\t      2. List CSV Files\n");
    printf("\t\t\t\t\t\t      3. Enter a Directory\n");
    printf("\t\t\t\t\t\t      4. Go Back to Parent Directory\n");
    printf("\t\t\t\t\t\t      5. Change Drive\n");
    printf("\t\t\t\t\t\t      6. Exit\n");
    printf("\t\t\t\t\t\t      Enter your choice: ");
    scanf("%d", &choice);
    if(choice==6)
    display_admin_menu();
    switch (choice) {
        case 1:
            listDirectories(currentPath);
            getchar();
            break;
        case 2:
            listCSVFiles(currentPath);
            break;
        case 3:
            if (changeDirectory(currentPath) == -1) {
                printf("\t\t\t\t\t\t\t Failed to change directory. Please try again.\n");
            }
            break;
        case 4:
            goBackDirectory(currentPath);
            break;
        case 5:
            driveSelection(currentPath);  // Recursive call to change drive
            break; // Prevent further menu actions after drive selection
//        case 6:
            //printf("\t\t\t\t\t\t Exiting the program.\n");
            // exit(0);
//            display_admin_menu();
        break;
        default:
            if(choice==6)
            menu_call();

            printf("\n\t\t\t\t\t\t\t\tInvalid choice. Please try again.\n");
            sleep(2);
            listDrives();
            break;
    }

    // After executing the choice, show the menu again recursively
    showMenu(currentPath);
}
```

changeDirectory:

- Prompt the user to enter the new path.
- Validate if the new path exists and is a directory:
    - If valid, update the current path to the new directory.
    - If invalid, print an error and return.

```
// Function to change the current directory
int changeDirectory(char *currentPath) {
    char newPath[max_path];
    printf("\t\t\t\t\t\t Enter the path to change to: ");
    scanf("%s", newPath);
/*

    DWORD attributes = GetFileAttributes(newPath);
    if (attribute == INVALID_FILE_ATTRIBUTES) {
        printf("\t\t\t\t\t\t Error: Path does not exist.\n");
        return -;
    }

*/



    DWORD attributes = GetFileAttributes(newPath);
    if (attributes == INVALID_FILE_ATTRIBUTES) {
        printf("\t\t\t\t\t\t Error: Path does not exist.\n");
        return -1;
    }

    if (!(attributes & FILE_ATTRIBUTE_DIRECTORY)) {
        printf("\t\t\t\t\t\t Error: Not a directory.\n");
        return -1;
    }

strncpy(currentPath, newPath, max_path);
    return 0;
}
```

goBackDirectory:

- Go back to the parent directory by removing the last part of the current path.
- If already at the root, reset the path to the root drive.

```
// GoBack to Parent Directory --------->Function to go back to the parent directory
void goBackDirectory(char *currentPath) {
    char *lastBackslash = strrchr(currentPath, '\\');
    if (lastBackslash != NULL && lastBackslash != currentPath + 2) {
        *lastBackslash = '\0';
    } else {
        currentPath[0] = '\0';  // Goback to the root of the drive
    }
}
///END PASSWORD RANDOM CREATOR
```

Pseudo Code for Current Session Details:

1. Function to Validate Required Columns in the Header

- **Input:** A string representing the header line of the CSV file.
- **Process:**
  - Loop through the required columns.
  - For each required column, check if it exists in the header line.
  - If any required column is missing, return **0** (invalid).
  - If all required columns are found, return **1** (valid).
- **Output:** Return **0** if any column is missing, or **1** if all columns are found.

2. Function to Parse and Print Configuration Data

- **Input:** A line of configuration data (comma-separated).
- **Process:**
  - Split the line into different parts using commas.
  - Extract and store:
    - Position of the election.
    - Number of candidates.
    - List of candidates.
    - Voter list path.
    - Path for verification.
    - Session time.
    - Configuration status.
  - Print each piece of extracted data in a readable format.
  - For candidates, split the candidate names by semicolons and print each one.
- **Output:** Print the parsed data in a user-friendly format.

3. Function to Read and Process the Configuration File

- **Input:** The filename of the configuration file.
- **Process:**
  - Open the configuration file.
  - If the file does not exist, display an error message and stop further processing.
  - Read the header line from the file.
    - If the header is invalid (missing required columns), print an error message and stop.
  - If the header is valid, read the next line of configuration data.
    - If data is found, parse and print it.

        - If no data row is found, print an error message.
    - Close the file after processing.
- **Output:** Print the configuration data or an error message if the file or data is invalid.

Full Pseudocode for Menu/Process
Main Menu (Session Handling)

- **Start:**
    - Prompt the user with a list of available options.
    - Let the user choose an option.
- **Choices:**

1. **List available drives**
    - Display a list of available drives (A, B, C, etc.).
    - Wait for the user to select a drive.

2. **List CSV files**
    - Ask the user to choose a directory.
    - List all CSV files in the selected directory.
    - Allow the user to choose a CSV file.
    3.
4. **Enter a directory**
    - Ask the user to input a directory path.
    - Change the current directory to the chosen one.
5. **Go back to the parent directory**
    - Move the current directory back to its parent folder.
6. **Change drive**
    - Allow the user to choose a new drive (e.g., C, D).
    - Update the current path accordingly.
7. **Exit**
    - Exit the program.

- **Repeat**: After each action, prompt the user with the main menu again until they select the "Exit" option.

```
// Function to list available drives
void listDrives() {
    DWORD drives = GetLogicalDrives();
   // printf("\t\t\t\t\t   ====================================\n");
    printf("\t\t\t\t\t    Available Drives:\n");
    for (char i = 'A'; i <= 'Z'; i++) {
        if (drives & (1 << (i - 'A'))) {
            printf("\t\t\t\t\t    Drive: %c:\n", i);
        }
    }
}
```

```c
void listCSVFiles(const char *path) {
printf("\n\t\t\t\t\t\t      CSV Files in %s:\n\t\t\t\t\t\t      ", path);

    WIN32_FIND_DATA findFileData;
char searchPath[max_path];
    snprintf(searchPath, max_path, "%s\\*.csv", path);   //print path krny ky liye

    HANDLE hFind = FindFirstFile(searchPath, &findFileData);

    if (hFind == INVALID_HANDLE_VALUE) {
        printf("\t\t\t\t\t\t  No CSV files found in the current directory.\n");
        sleep(2); // screen stop for 2sec here
        return;
    }

    int count = 0;
    do {
        printf("\t\t\t\t\t\t      %d. %s\n", ++count, findFileData.cFileName);
    } while (FindNextFile(hFind, &findFileData) != 0);
//    FindClose(hFind, FindData);

    FindClose(hFind);
    if (count == 0) {
    printf("\t\t\t\t\tNo CSV files found in the current directory.\n");
sleep(2);
        return;
    }

    // Allow user to select a file
int fileChoice;
    printf("\t\t\t\t\t\t      Enter the number of the CSV file to modify: ");
    scanf("%d", &fileChoice);
    sleep(2);
hFind = FindFirstFile(searchPath, &findFileData);
    for (int i = 1; i < fileChoice && FindNextFile(hFind, &findFileData); i++);

    if (fileChoice <= 0 || findFileData.cFileName[0] == '\0') {
        printf("\t\t\t\t\t\t Invalid selection.\n");
        sleep(2);
        FindClose(hFind); //closing the file
        return;
    }

    char selectedFile[max_path]; //selected file path
    snprintf(selectedFile, max_path, "%s\\%s", path, findFileData.cFileName);

    // Add password column and save to the same directory
    if (addPasswordColumnToCSV(selectedFile) == 0) {
printf("\t\t\t\t\t\tPassword column added successfully to %s.\n", selectedFile);
        sleep(2);
    }

    FindClose(hFind);
}
```

Pseudo Code for Change Admin Password:

Function encrypt_text_1(text):
    FOR each character in text:
        XOR character with KEY
    END FOR

Function write_data_2(user, pass, q1, q2, q3):
  Open FILE_NAME in write mode
  IF file not opened successfully:
    Print "Error creating file"
    RETURN

  Encrypt user, pass, q1, q2, q3 using encrypt_text_1
  Write encrypted data to file
  Close the file

Function read_data_3(user, pass, q1, q2, q3):
  Open FILE_NAME in read mode
  IF file not opened:
    RETURN 0 (File not found)

  Read encrypted user, pass, q1, q2, q3 from file
  Decrypt each data field using encrypt_text_1
  Close the file
  RETURN 1 (Data successfully read)

Function change_password_4():
  Declare old_pass, new_pass, choice variables
  Declare stored_user, stored_pass, q1, q2, q3

  IF read_data_3 fails:
    Print "Data file not found"
    RETURN

  WHILE TRUE:
    Display admin menu
    Prompt for current password (old_pass)
    IF old_pass matches stored_pass:
      Prompt for new password (new_pass)
      Update stored password and write new data to file
      Print "Password changed successfully"
      Wait for 2 seconds
      Call menu_call
      BREAK
    ELSE:
      Print "Current password does not match"
      Ask user if they want to try again (Y/N)
      IF user chooses N or n:
        Print "Exiting without changing the password"
        Wait for 2 seconds
        Call menu_call
        BREAK
END Function

```c
// Handles changing -----> user's password
void change_password_4() {
    char old_pass[100], new_pass[100], choice;
    char stored_user[100], stored_pass[100], q1[100], q2[100], q3[100];

    // Check-----AS data file exists and read current user data
    if (!read_data_3(stored_user, stored_pass, q1, q2, q3)) {
        printf("\t\t\t\t\t Data file not found. Please contact the administrator.\n");
        return;
    }

    while (1) {
        display_admin_menu();
        printf("\t\t\t\t\t Enter your current password: ");
        scanf("%s", old_pass);

        // Verify the OLDD password
        if (strcmp(old_pass, stored_pass) == 0) {
            printf("\t\t\t\t\t Enter a new password: ");
            scanf("%s", new_pass);

            // Update ----------> password and write back the data
            write_data_2(stored_user, new_pass, q1, q2, q3);
            printf("\t\t\t\t\t Password changed successfully...\n");
            sleep(2);
            menu_call();
            break;
        } else {
        printf("\t\t\t\t\t Current password does not match.\n");
        printf("\t\t\t\t\t Do you want to try again? (Y/N): ");
        scanf(" %c", &choice);

            if (choice == 'N' || choice == 'n') {
                printf("\t\t\t\t\t Exiting without changing the password...\n");
                sleep(2);
                menu_call();
                break;
            }
    }
    }
}
```

Pseudo Code for Exit:
Go to display_main_menu.

Pseudo Code for Start Voting:

Display Voting Menu

- Clear screen.
- Show header and list available voting positions from "config.csv".
- Check "flag.txt"; update status if needed.
- Start login process.

2. Login (main_login_voters)

- Load voting duration from "config.csv".
- Record current time as start_time.
- Create "time_log.txt" if it doesn't exist to log voting period.

3. Voting Loop

- Repeat until voting session ends:
    - **Check Remaining Time**
        - Calculate remaining time (hours, minutes) since start_time.
        - Write remaining time to "time_log.txt".
        - End session if time is up and return to main menu.
    - **Voting Process**
        - Load candidates from "config.csv".
        - Prompt user for ID and password.
        - Verify credentials and voting eligibility.
        - If user already voted or invalid, return to main menu.
        - Show list of candidates, prompt user to select.
        - **Record Vote**
            - If valid choice, update user's vote in "copy_of_candidates.csv".
            - Confirm vote submission.

4. Helper Functions

- **Trim String**: Remove extra spaces around string.
- **Load Voting Duration**: Retrieve hours, minutes from "config.csv".
- **Initialize Time Log**: Create "time_log.txt" if it doesn't exist.
- **Update Time Log**: Calculate remaining time, write to "time_log.txt".
- **Load Candidates**: Read candidate names from "config.csv".
- **Count IDs**: Get total valid IDs in "copy_of_candidates.csv".
- **Validate User**: Check if ID and password match, and if user hasn't voted.
- **Record Vote**: Update user's vote in "copy_of_candidates.csv".

5. End Voting Session

- If all users have voted or time is up, return to the main menu.

```c
void main_login_voters() {
    int allowed_hours, allowed_minutes;
    if (!load_voting_duration(&allowed_hours, &allowed_minutes)) {
        printf("\t\t\t\t\t   Error loading voting duration from config.\n");
        return;
    }

    time_t start_time = time(NULL);
    initialize_time_log(allowed_hours, allowed_minutes);

    while (1) {
        update_remaining_time(start_time, allowed_hours, allowed_minutes);
        vote_in_election(start_time, allowed_hours, allowed_minutes);
    }
}
///Result----> previsous result and result creation

int main_result() {
    int choice;
    do {
        system("cls");
        display_admin_menu();
        printf("\n\t\t\t\t\t   1. Create Voting Result\n");
        printf("\t\t\t\t\t   2. See Results\n");
        printf("\t\t\t\t\t   3. Exit\n");
        printf("\t\t\t\t\t   Enter your choice: ");

        if (scanf("%d", &choice) != 1) {
            printf("\t\t\t\t\t   Invalid choice. Try again.\n");
            while (getchar() != '\n');
            sleep(2);
            continue;
        }
```