**NBKR INSTITUTE OF SCIENCE AND TECHNOLOGY**

**BASIC E-VOTING SYSTEM**

**COURSE:** DATA STRUCTURES

**DEPARTMENT:** COMPUTER SCIENCE ENGINEERING

**SECTION:** E

**YEAR:** I

**SEMESTER:** II

**DATE:** 01 MAY 2025

**SUBMITTED TO:** ASHOK SELVAKUMAR E

**SUBMITTED BY:**

S.YASASWINI(24KB1A05KB)

T.LAHARI(24KB1A05KU)

U.PALLAVI(24KB1A05LW)

TANUSHREE(24KB1A05KQ)

**ACKNOWLEDGEMENT**

We, the project team, would like to express our sincere gratitude to everyone who supported and guided us throughout the successful completion of our project titled **"Basic E-Voting System."**

First and foremost, we extend our heartfelt thanks to our project guide, **Ashok Selva Kumar E**, for their invaluable guidance, encouragement, and constant support. Their expertise and insights have been instrumental in the successful development of this system.

We are also grateful to **NBKR INSTITUTE OF SCIENCE AND TECHNOLOGY** especially the faculty and staff of the **Computer Science** department, and **Bytexl** platform for providing us with the resources and academic foundation necessary for this project.

This project allowed us to practically apply our knowledge of the **C programming language**, along with key concepts from **Data Structures**, in designing and implementing a basic electronic voting system. It has been a valuable learning experience in both technical and collaborative aspects.

We would also like to thank our peers, friends, and families for their continuous encouragement and moral support during the course of this work.

**ABSTRACT**

The **Basic E-Voting System** is a console-based application developed in C using fundamental data structures. It allows candidate registration, vote casting using unique Voter IDs, and vote tallying. The system ensures each voter can vote only once and displays the winner based on the highest votes. Arrays and structures are used to manage candidates and voter data. This project demonstrates how core programming concepts can be applied to create simple and efficient digital voting solutions.

**INTRODUCTION**

The **Basic E-Voting System** is a simple C-based project designed to simulate the process of electronic voting using fundamental data structures. It allows candidate registration, vote casting with unique voter IDs, vote counting, and winner declaration. The idea was chosen to explore how core programming concepts can be applied to solve real-world problems like digital elections. This project helps understand how to manage data securely and efficiently. It also highlights the importance of fairness and accuracy in vote-based systems.

**OBJECTIVE**

The main objective of this project is to develop a simple and functional electronic voting system using the C programming language and basic data structures. The system aims to:

* Simulate a real-world voting process digitally.
* Allow candidate registration and secure vote casting.
* Ensure that each voter can vote only once using a unique Voter ID.
* Accurately count votes and declare the winner.
* Apply programming concepts like arrays and structures in a practical application.

**SYSTEM REQUIREMENTS**

* **SOFTWARE:**

Operating System: Windows, Linux.

Compiler/IDE:

C Compiler: Dev c++, github,

IDE: Code::Blocks, Visual Studio Code, or Turbo C

Libraries: Standard C libraries (stdio.h, string.h, etc.)

* **HARDWARE:**

Processor: Intel Pentium or higher / Equivalent AMD processor

RAM: Minimum 2 GB

Hard Disk Space: 100 MB of free space

Display: Minimum 1024x768 resolution

**METHODOLOGY**

The development of the Basic E-Voting System followed a step-by-step approach using the **C programming language** and basic **data structures** like arrays and structures. The process is outlined below:

1. **Problem Definition**:  
   The aim was to create a simple electronic voting system that allows candidate registration, secure voting, vote counting, and winner declaration.
2. **System Design**:  
   The system was designed to use a **structure** to store candidate details (name and votes), and an **array** to track which voters have already voted (to avoid duplicate voting).
3. **Implementation**:
   * Candidate registration is handled by storing names and initializing their vote count to zero.
   * Voting is done by checking the voter ID and updating the vote count of the selected candidate.
   * Vote counts are displayed and the winner is determined by comparing vote totals.
4. **Testing**:  
   The system was tested with various inputs to ensure that:
   * Voters can vote only once.
   * Votes are correctly counted.
   * Invalid inputs (like wrong voter IDs or wrong candidate choices) are handled properly.
5. **Execution**:  
   The program was executed in a C compiler (like GCC or Turbo C), and all functionalities worked as intended in a command-line environment.

**PROJECT DESCRIPTION**

**Problem Statement**

Traditional voting systems often face challenges such as manual errors, long processing times, duplicate voting, and lack of security. These issues reduce the efficiency and trust in the election process, especially in environments where large numbers of voters are involved.

**Proposed Solution**

To overcome these issues, we propose a Basic E-Voting System developed using the C programming language and data structures. This system provides a simple, efficient, and secure way to conduct digital voting through a console-based application. It ensures that each voter votes only once and automatically counts and displays results without human error.

**Key Features**

* **Candidate Registration**: Allows admin to register multiple candidates.
* **Secure Voting**: Accepts votes using unique voter IDs and prevents duplicate voting.
* **Vote Counting**: Automatically counts votes for each candidate.
* **Winner Declaration**: Displays the candidate with the highest number of votes.
* **User-Friendly Interface**: Simple command-line interaction for ease of use.

**ALGORITHM**

1. **Start the program.**
2. **Initialize**:
   * An array of candidate structures (to store names and votes).
   * An array to track if a voter has already voted.
3. **Display the main menu** with options:
   * Register Candidate
   * Vote
   * Count Votes
   * Declare Winner
   * Exit
4. **If the user selects “Register Candidate”**:
   * Check if the maximum limit is reached.
   * Take the candidate's name as input.
   * Add the candidate to the list and initialize their votes to 0.
5. **If the user selects “Vote”**:
   * Prompt for voter ID.
   * Check if the ID is valid (between 0 and MAX\_VOTERS).
   * Check if the voter has already voted.
   * Display the list of registered candidates.
   * Take vote input and increment the vote count for the selected candidate.
   * Mark the voter as having voted.
6. **If the user selects “Count Votes”**:
   * Display the total votes for each candidate.
7. **If the user selects “Declare Winner”**:
   * Find the candidate with the maximum votes.
   * Display the winner's name and vote count.
8. **If the user selects “Exit”**:
   * End the program.
9. **Repeat** steps 3–8 until the user chooses to exit.
10. **Stop the program.**

**PROGRAM CODE**

#include <stdio.h>

#include <string.h>

#define MAX\_CANDIDATES 10

#define MAX\_VOTERS 10000

// Structure to store candidate information

typedef struct {

char name[100];

int votes;

} Candidate;

Candidate candidates[MAX\_CANDIDATES];

int candidate\_count = 0;

int voted[MAX\_VOTERS] = {0}; // To track whether a voter ID has already voted

void register\_candidate() {

if (candidate\_count >= MAX\_CANDIDATES) {

printf("Cannot register more candidates.\n");

return;

}

printf("\nEnter candidate name: ");

scanf("%s", candidates[candidate\_count].name);

candidates[candidate\_count].votes = 0;

candidate\_count++;

printf("Candidate registered.\n");

}

void vote() {

int voter\_id;

printf("\nEnter Voter ID (0-9999): ");

scanf("%d", &voter\_id);

if (voter\_id < 0 || voter\_id >= MAX\_VOTERS) {

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

return;

}

if (voted[voter\_id]) {

printf("You have already voted.\n");

return;

}

if (candidate\_count == 0) {

printf("No candidates registered yet.\n");

return;

}

printf("Candidates:\n");

for (int i = 0; i < candidate\_count; i++) {

printf("%d. %s\n", i + 1, candidates[i].name);

}

int choice;

printf("Enter your choice (number): ");

scanf("%d", &choice);

if (choice < 1 || choice > candidate\_count) {

printf("Invalid choice.\n");

return;

}

candidates[choice - 1].votes++;

voted[voter\_id] = 1;

printf("Vote recorded.\n");

}

void count\_votes() {

printf("\nVote Count:\n");

for (int i = 0; i < candidate\_count; i++) {

printf("%s: %d votes\n", candidates[i].name, candidates[i].votes);

}

}

void declare\_winner() {

if (candidate\_count == 0) {

printf("No candidates to declare winner.\n");

return;

}

int max\_votes = -1;

int winner\_index = -1;

for (int i = 0; i < candidate\_count; i++) {

if (candidates[i].votes > max\_votes) {

max\_votes = candidates[i].votes;

winner\_index = i;

}

}

printf("\nWinner: %s with %d votes\n", candidates[winner\_index].name, candidates[winner\_index].votes);

}

int main() {

int option;

while (1) {

printf("\n--- E-Voting Menu ---\n");

printf("1. Register Candidate\n");

printf("2. Vote\n");

printf("3. Count Votes\n");

printf("4. Declare Winner\n");

printf("5. Exit\n");

printf("Choose an option: ");

scanf("%d", &option);

switch (option) {

case 1:

register\_candidate();

break;

case 2:

vote();

break;

case 3:

count\_votes();

break;

case 4:

declare\_winner();

break;

case 5:

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

return 0;

default:

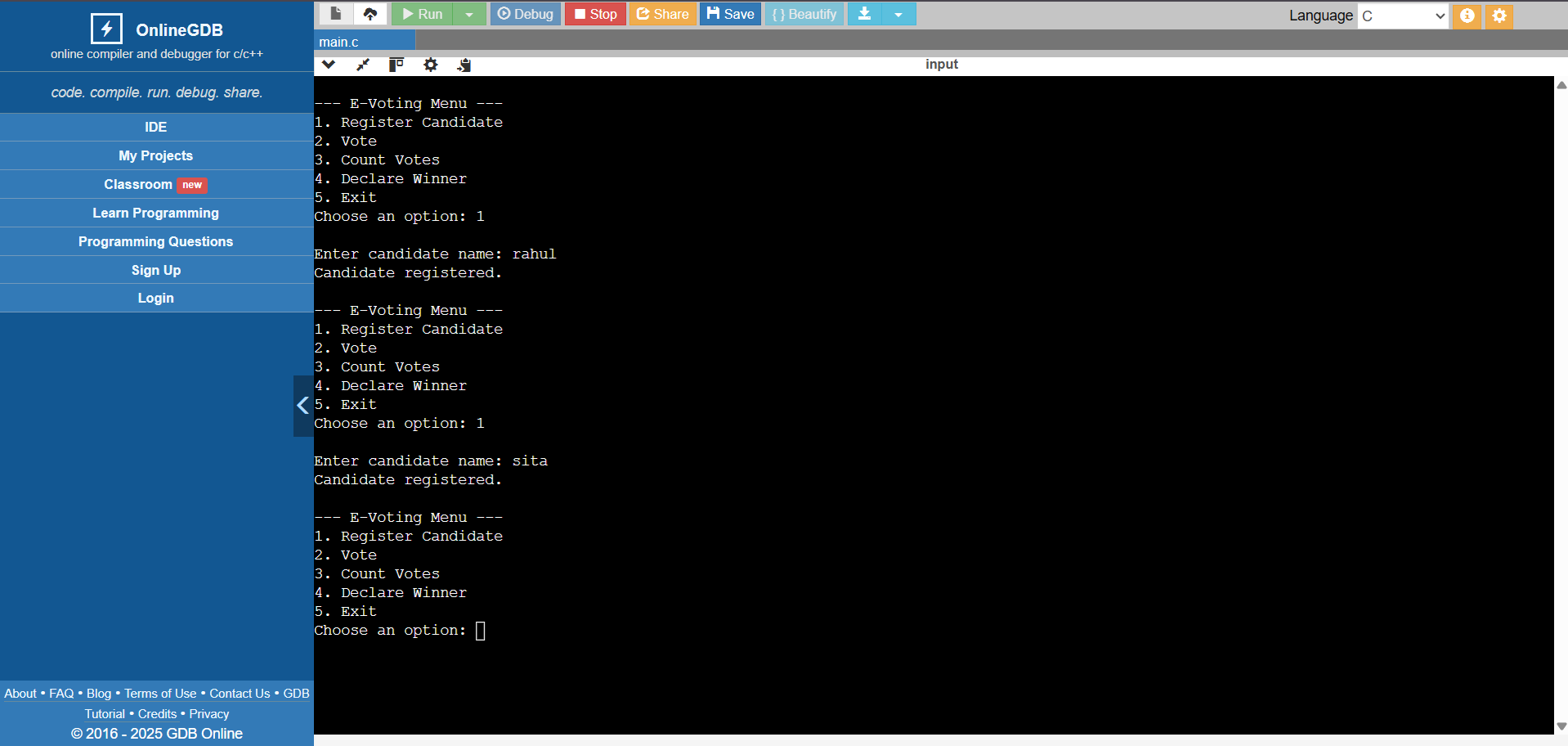
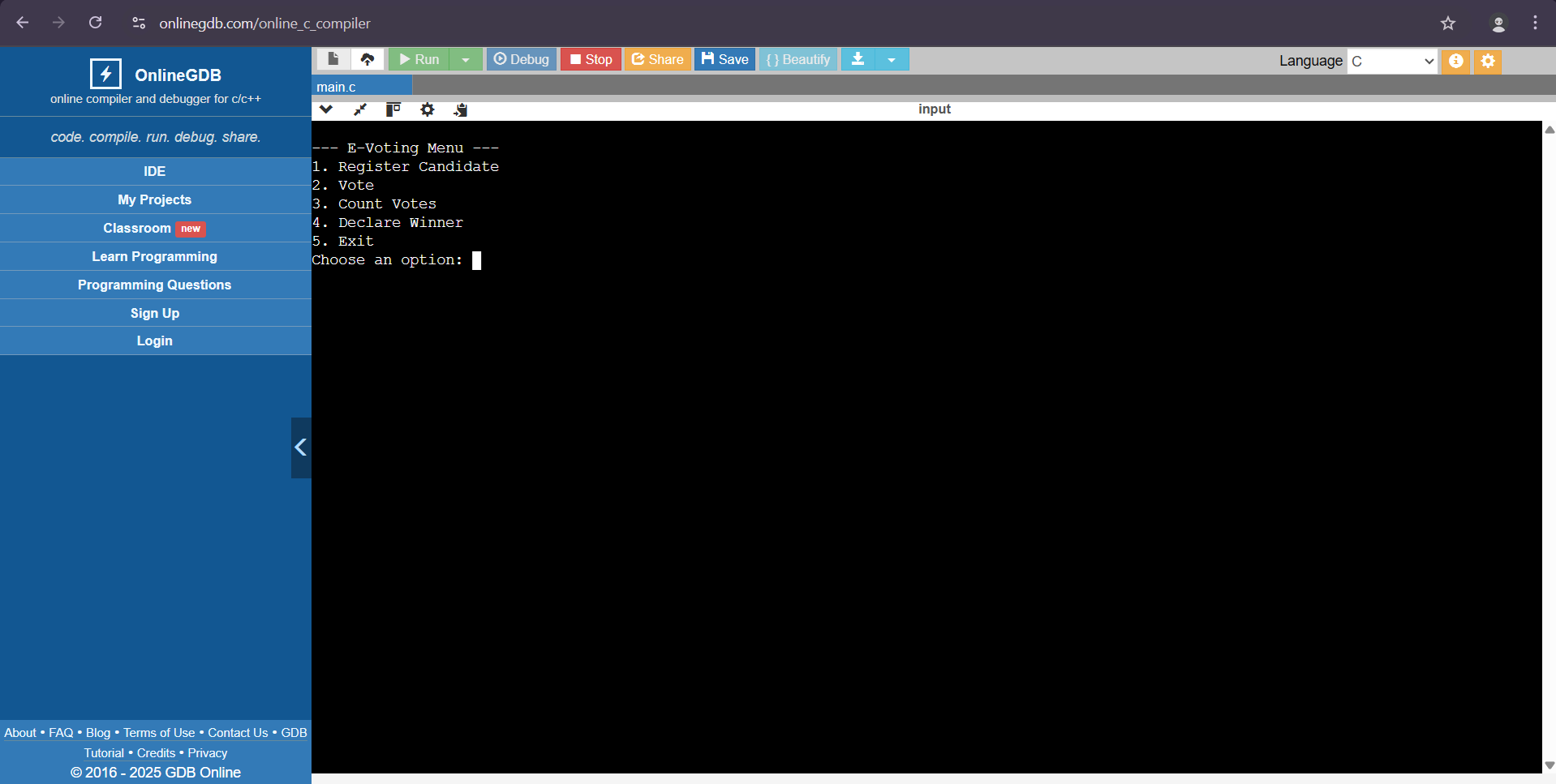
printf("Invalid option. Please try again.\n");

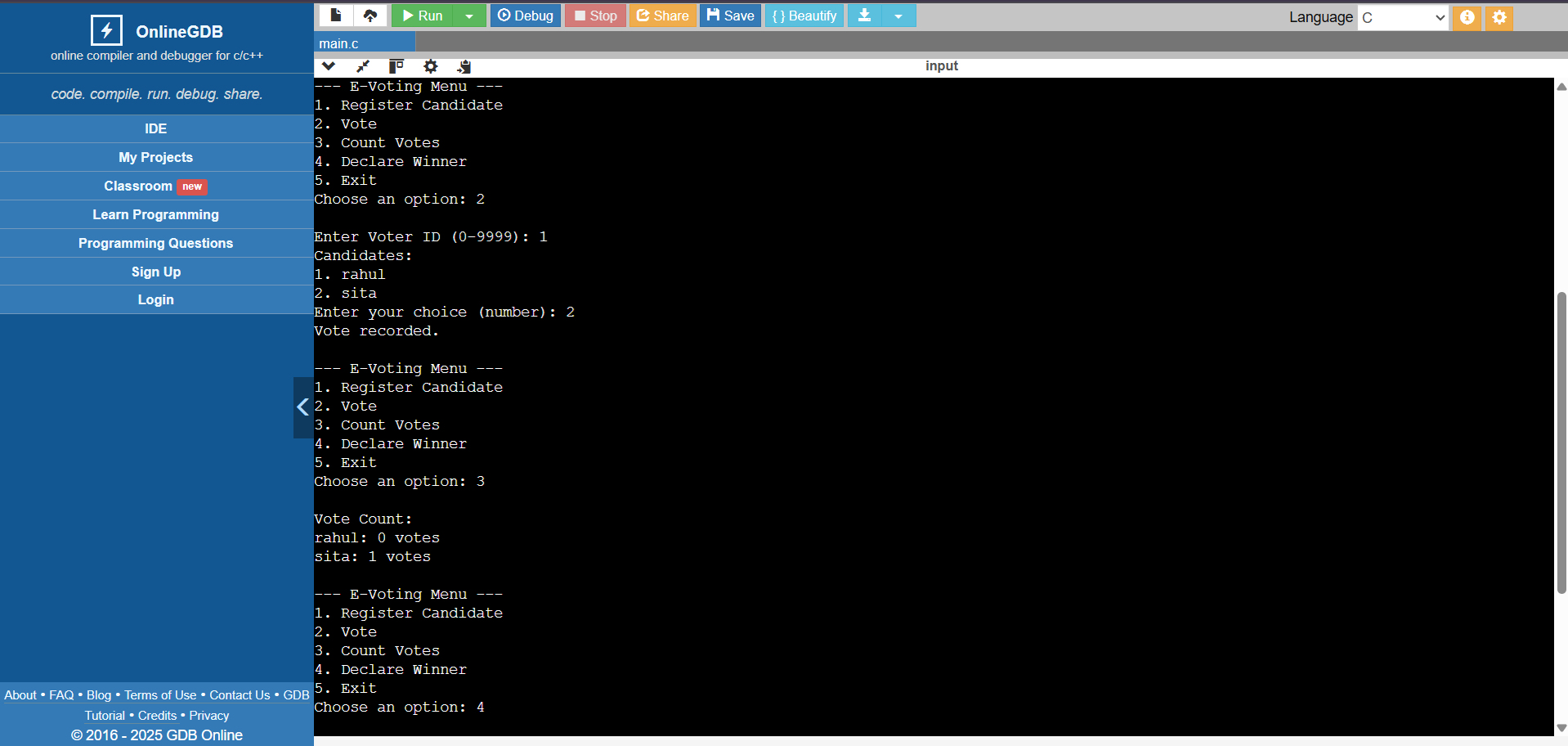
}

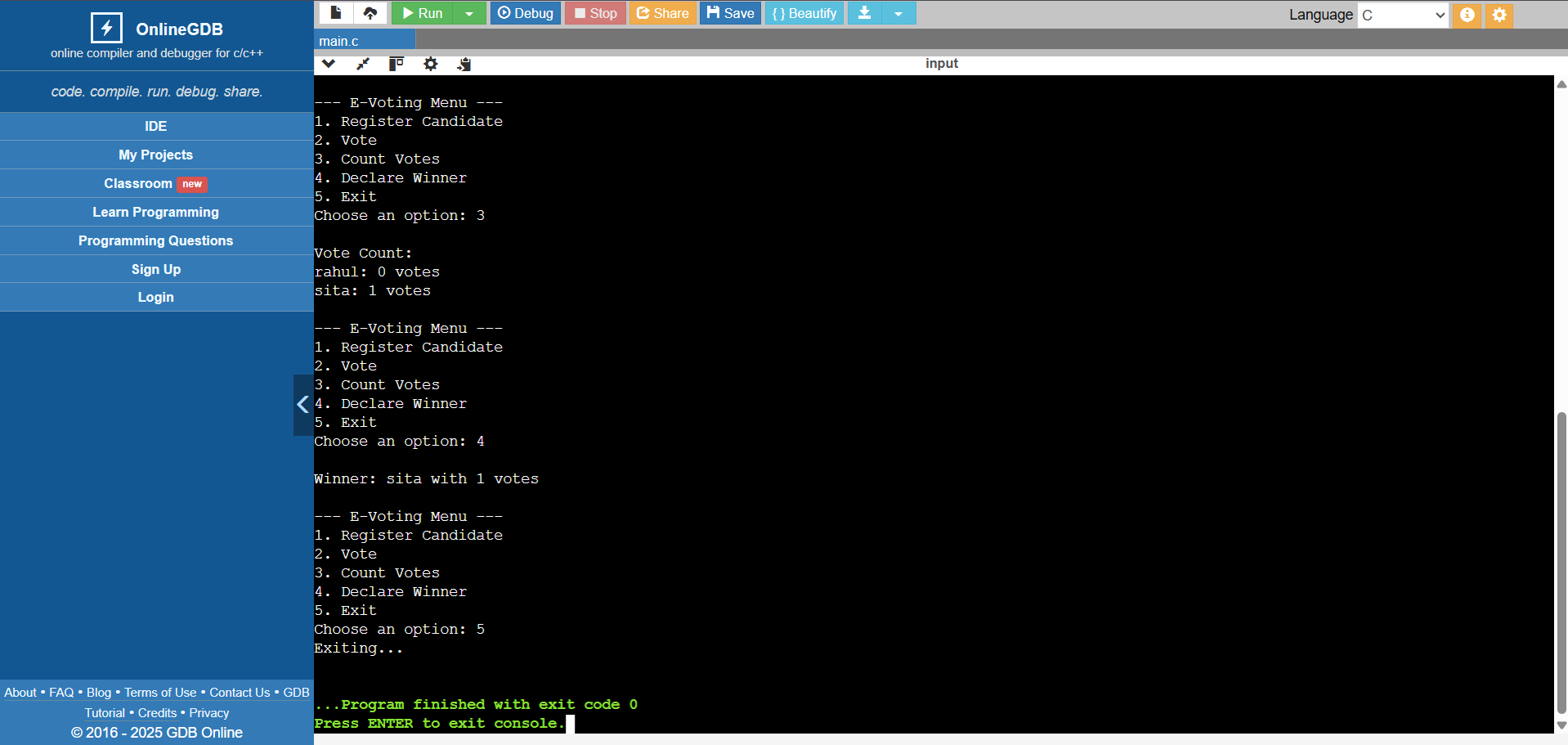
}

return 0;

}

**OUTPUT **

****

****

**TESTING AND VALIDATION**

1. Register Candidate Test

* Valid Candidate Registration: Test registering up to MAX\_CANDIDATES (10). Ensure proper registration and initialization of votes to 0.
* Exceeding Candidate Limit: Try registering more than 10 candidates. Ensure the system prevents it with an error message.

2. Voting Test

* Valid Vote: After registering candidates, test voting with a valid Voter ID. Ensure the vote is recorded correctly.
* Invalid Voter ID: Test voting with Voter IDs out of bounds (e.g., -1, 10000). Ensure an "Invalid Voter ID" message.
* Double Voting: Attempt to vote twice with the same Voter ID. Ensure the system prevents multiple votes.
* No Candidates Registered: Try voting without any registered candidates. Ensure the system informs that no candidates are available.
* Invalid Candidate Choice: Select an invalid candidate number when voting. Ensure an "Invalid choice" message.

3. Count Votes Test

* After Voting: Count votes after voting occurs. Ensure correct vote counts for each candidate.
* Without Votes: Count votes with no votes cast. Ensure all candidates show 0 votes.

4. Declare Winner Test

* Valid Winner Declaration: After voting, declare the winner. Ensure the candidate with the most votes wins.
* Tie Scenario: Test with a tie in votes. Ensure the program handles ties properly.
* No Votes Cast: Declare the winner when no votes have been cast. Ensure a valid message is shown.

5. Exit Test

* Exit Program: Choose the "Exit" option and ensure the program closes without errors.

Edge Cases

* Voter ID Boundaries: Test with boundary Voter IDs (0, 9999).
* Maximum Limits: Test with MAX\_CANDIDATES (10) and MAX\_VOTERS (10000) to ensure system stability.

**LIMITATIONS**

1. **Limited Candidates and Voters**: It supports up to 10 candidates and 10,000 voters, which may not scale for larger elections.
2. **No Voter Authentication**: No real security for voter identity, making it vulnerable to fraud.
3. **Single Vote Per Voter**: Voters cannot modify or retract votes once cast.
4. **No Real-Time Results**: Vote count updates only after manually choosing the "Count Votes" option.
5. **Basic UI**: The text-based interface is not user-friendly for larger elections.
6. **No Data Persistence**: Data is lost after the program ends; no storage solution is implemented.
7. **No Tie Handling**: The system doesn't handle ties properly.
8. **Scalability Issues**: Cannot handle large numbers of voters or candidates efficiently.
9. **Security Risks**: No encryption or secure communication for votes.
10. **Lack of Privacy**: Votes are not anonymized, which could lead to privacy concerns.

These limitations highlight areas for improvement in security, scalability, and usability.

**FUTURE ENHANCEMENT**

* **Secure Voter Login** – Add authentication (e.g., OTP, ID verification).
* **Data Storage** – Use a database to save candidates and votes.
* **Real-Time Results** – Show live vote counts.
* **GUI Support** – Add a graphical user interface for ease of use.
* **Encryption** – Secure vote data with encryption.
* **Voter Privacy** – Make votes anonymous.
* **Tie Handling** – Add logic to resolve vote ties.
* **Mobile Access** – Make system usable on smartphones.
* **Audit Logs** – Keep records of votes for verification.
* **Scalability** – Optimize for more voters and candidates.

**CONCLUSION**

The e-voting system provides a simple and efficient way to conduct digital elections. It includes features like candidate registration, secure voting, vote counting, and winner declaration. The system ensures one vote per voter and prevents duplicate voting. Compared to manual voting, it significantly reduces time, effort, and errors. While suitable for small-scale use, future enhancements like authentication, data storage, and a better UI can improve its security, scalability, and usability for larger applications.

**REFERENCES**

1. Algorithms and Data Structures: The Basic Toolbox by Kurt Mehlhorn and Peter

Sanders

2. C Data Structures and Algorithms by Alfred V. Aho, Jeffrey D. Ullman, and John E.

Hopcroft

3. Problem Solving with Algorithms and Data Structures" by Brad Miller and David

Ranum

4. Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson, Ronald L.

Rivest, and Clifford Stein

5 . Websites: <https://www.programiz.com>