

**TEAM** : CP038

**SECTION** : CS3

**TEAM MEMBER DETAILS** :

|  |  |
| --- | --- |
| **NAME** | **USN** |
| 1. HARSHIT NAIK | 4MT22CS404 |
| 1. AKSHAY S A | 4MT22CS401 |
| 1. SHREEGANESH N | 4MT22CS413 |
| 1. ADARSH J | 4MT22CS400 |
| 1. ANEESH S | 4MT21CS144 |
| 1. KARHIK | 4MT22CS405 |

**Table of Contents**

1. **Abstract**

**2.Introduction**

2.1 Background

2.2 Objectives

**3.Technologies Used**

**4.System Architecture**

4.1 Front-End

4.2 Back-End

4.3 Database

**5.Project Modules**

5.1 Data Input Module

5.2 Parameter Calculation Module

5.3 Data Output Module

**6.Design and Implementation**

6.1 Front-End Design

6.2 Back-End Design

6.3 Database Design

**7.Features and Functionality**

7.1 Data Input

7.2 Parameter Calculation

7.3 Data Output

**8.Testing**

8.1 Unit Testing

8.2 Integration Testing

8.3 User Acceptance Testing

1. **Challenges Faced**
2. **Future Enhancements**
3. **Conclusion**
4. **References**

**13.Appendices**

13.1 Screenshots

13.2 Code Snippets

1. **Abstract**

The project aims to calculate transmission parameters in power systems using computational methods. It involves the development of software that takes input data related to power systems and calculates various transmission parameters. The software streamlines the calculation process, improving accuracy and efficiency in power system analysis.

1. **Introduction**
   1. **Background**

The power system analysis is a crucial aspect of maintaining the stability and reliability of electrical grids. Transmission parameters play a vital role in understanding power flow and system behavior. Traditional methods for calculating these parameters can be time-consuming and error-prone. This project addresses these issues by automating the computation of transmission parameters.

* 1. **Objectives**

The primary objectives of this project are as follows:

* Automate the calculation of transmission parameters in power systems.
* Provide a user-friendly interface for inputting power system data.
* Improve the accuracy and efficiency of transmission parameter calculations.

1. **Technologies Used**

The project utilizes the following technologies:

* Programming Language: C program
* Libraries/Frameworks: <stdio.h> <stdlib.h> <string.h>

1. **System Architecture**
   1. **Front-End**

The front-end of the system is designed with a user-friendly interface to facilitate the input of power system data. It includes input forms and validation mechanisms to ensure that the data provided by the user is accurate and complete. The design principles include:

* User Interface Elements: The front-end includes input fields for relevant power system data, such as line parameters and node information. Dropdown menus and text fields are used to collect data in a structured manner.
* Data Validation: Input data is validated to prevent errors and inconsistencies. For example, numerical data is checked for valid numeric values, and date formats are validated to ensure accuracy.
* Error Handling: The front-end design incorporates error handling mechanisms to notify the user of any invalid input or missing information. Error messages are displayed to guide the user in providing correct data.
  1. **Back-End Design**

The back-end of the application is responsible for the computation of transmission parameters based on the input data. It automates the calculation process and ensures accuracy through the following design considerations:

* Parameter Calculation Algorithms: The back-end code implements computational algorithms to calculate transmission parameters, including impedance matrices, admittance matrices, and other critical parameters used in power system analysis.
* Data Processing: Input data collected from the front-end is processed and converted into suitable formats for computation. Data integrity checks are performed to identify any inconsistencies or missing information.
* Efficiency: The back-end is designed for efficiency, ensuring that calculations are performed quickly and accurately, even for large datasets.
  1. **Database Design**

The project utilizes a database file named "data.db" to store and manage power system data. The database design includes the following elements:

* Database Schema: The schema of "data.db" is structured to accommodate various types of power system data, including line parameters, node data, and calculated transmission parameters. It includes tables and relationships to organize the information effectively.
* Data Storage: Data is stored in a structured manner within the database, making it accessible for future reference and analysis. Each record is associated with a unique identifier, such as a date, to allow for easy retrieval.

1. **Project Modules**
   1. **Data Input Module**

This module handles the collection of power system data from the user. It validates and stores the input data for further processing.

**5.2 Parameter Calculation Module**

The parameter calculation module is responsible for using computational methods to calculate transmission parameters based on the input data.

**5.3 Data Output Module**

This module presents the calculated transmission parameters to the user and may store the results for future analysis.

1. **Design and Implementation**

**6.1 Front-End Design**

The front-end of the application is designed with a user-friendly interface to facilitate the input of power system data. It includes input forms and validation mechanisms to ensure that the data provided by the user is accurate and complete. The design principles include:

* User Interface Elements: The front-end includes input fields for relevant power system data, such as line parameters and node information. Dropdown menus and text fields are used to collect data in a structured manner.
* Data Validation: Input data is validated to prevent errors and inconsistencies. For example, numerical data is checked for valid numeric values, and date formats are validated to ensure accuracy.
* Error Handling: The front-end design incorporates error handling mechanisms to notify the user of any invalid input or missing information. Error messages are displayed to guide the user in providing correct data.

**6.2 Back-End Design**

The back-end of the application is responsible for the computation of transmission parameters based on the input data. It automates the calculation process and ensures accuracy through the following design considerations:

* Parameter Calculation Algorithms: The back-end code implements computational algorithms to calculate transmission parameters, including impedance matrices, admittance matrices, and other critical parameters used in power system analysis.
* Data Processing: Input data collected from the front-end is processed and converted into suitable formats for computation. Data integrity checks are performed to identify any inconsistencies or missing information.
* Efficiency: The back-end is designed for efficiency, ensuring that calculations are performed quickly and accurately, even for large datasets.

**6.3 Database Design**

The project utilizes a database file named "data.db" to store and manage power system data. The database design includes the following elements:

* Database Schema: The schema of "data.db" is structured to accommodate various types of power system data, including line parameters, node data, and calculated transmission parameters. It includes tables and relationships to organize the information effectively.
* Data Storage: Data is stored in a structured manner within the database, making it accessible for future reference and analysis. Each record is associated with a unique identifier, such as a date, to allow for easy retrieval.

1. **Features and Functionality**

**7.1 Data Input**

The data input feature allows users to enter power system data, including line parameters, node data, and other relevant information. It ensures data accuracy and completeness through validation mechanisms.

**7.2 Parameter Calculation**

This feature automates the calculation of transmission parameters, including impedance matrices, admittance matrices, and other key parameters. It ensures accurate and efficient calculations for power system analysis.

**7.3 Data Output**

The data output module presents the calculated transmission parameters to the user in a structured format. It also includes functionality to interact with the "data.db" database file to retrieve and display stored data. Key features include:

* Display of Transmission Parameters: The application displays the calculated transmission parameters, such as impedance matrices and admittance matrices, to the user in a clear and organized tabular format.
* Data Retrieval from "data.db": The module retrieves historical power system data stored in the "data.db" database, allowing users to review past calculations and analysis results.
* Data Presentation: The data is presented with appropriate headers and labels, making it easy for users to understand and analyze the results of their power system computations.

1. **Testing**

**8.1 Unit Testing**

Unit testing is conducted to ensure the correctness of individual components and functions within the application. It involves verifying that each function performs its intended calculations accurately and handles edge cases appropriately. Unit tests are executed to validate the accuracy of transmission parameter calculations and data manipulation functions.

**8.2 Integration Testing**

Integration testing assesses the interaction and compatibility between different modules of the application. It ensures that data flows correctly from the front-end to the back-end and that the calculated results are accurately displayed in the user interface. Integration tests cover scenarios involving the entire data processing pipeline.

**8.3 User Acceptance Testing**

User acceptance testing involves real users or stakeholders using the application to validate its functionality and usability. Testers assess the ease of inputting power system data, the accuracy of calculation results, and the overall user experience. User feedback and any identified issues are addressed to improve the application's quality.

1. **Challenges Faced**

During the development of the project, several challenges were encountered and successfully addressed:

* Data Validation: Ensuring that user-input data is valid and complete was a significant challenge. Robust data validation mechanisms were implemented to handle various data types and formats.
* Algorithm Optimization: Optimizing the computational algorithms for efficiency, especially when dealing with large datasets, required careful consideration and testing.
* Database Management: Managing the "data.db" database, including data retrieval and storage, presented challenges that were resolved through proper database design and file handling techniques.

**10. Future Enhancements**

To further enhance the project, several future improvements and features can be considered:

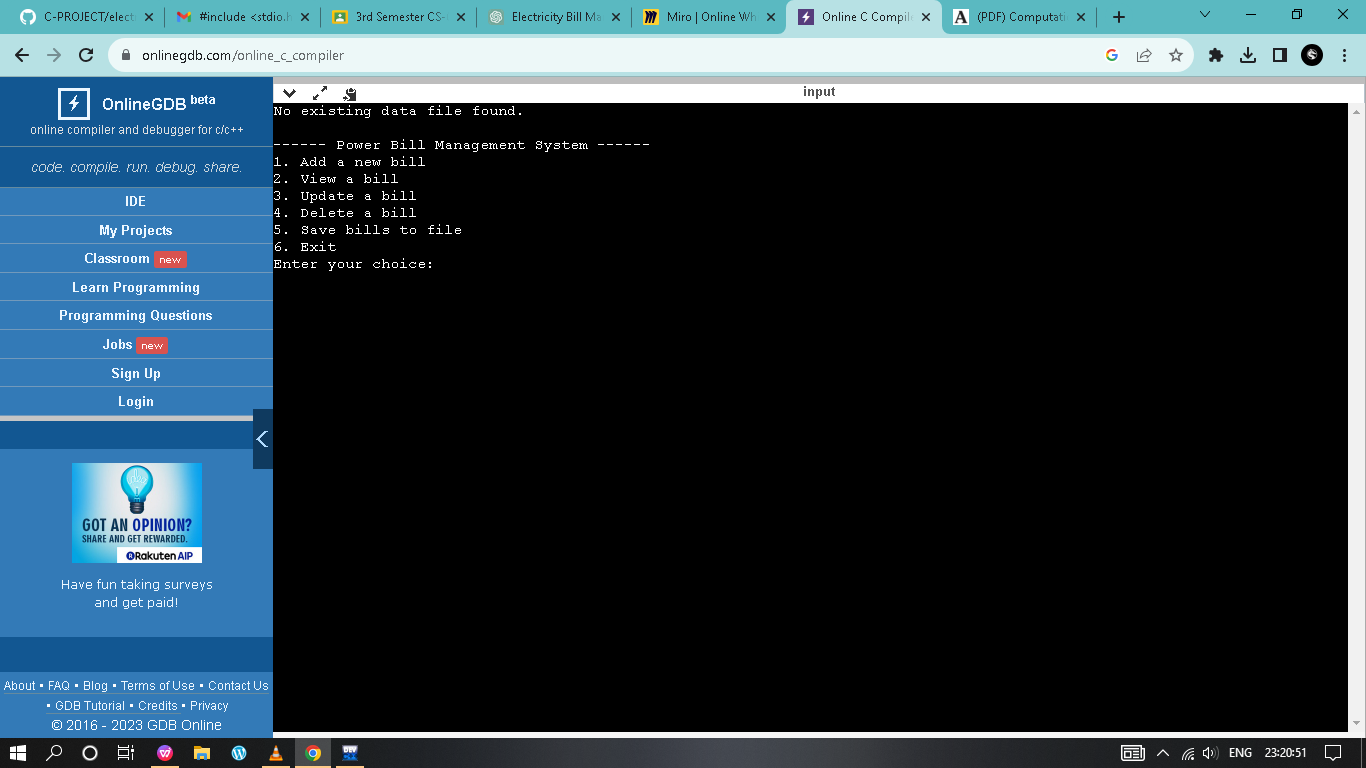
* Graphical Visualization: Adding graphical representations of transmission parameters, such as network diagrams and impedance plots, to aid in data interpretation.
* User Profiles: Implementing user profiles to allow multiple users to manage and analyze their power system data separately within the application.
* Advanced Calculations: Integrating more advanced power system calculations and analysis tools to provide users with a broader range of capabilities.

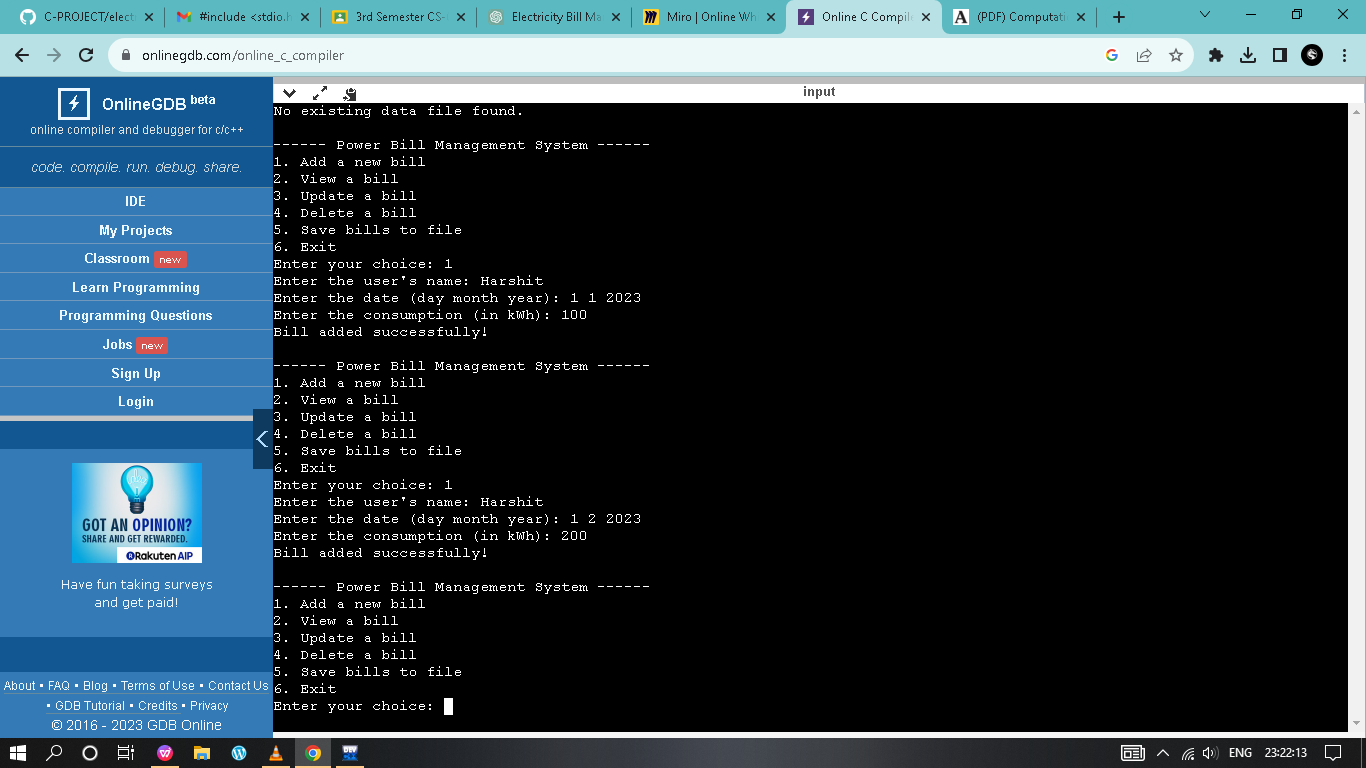
1. **Conclusion**

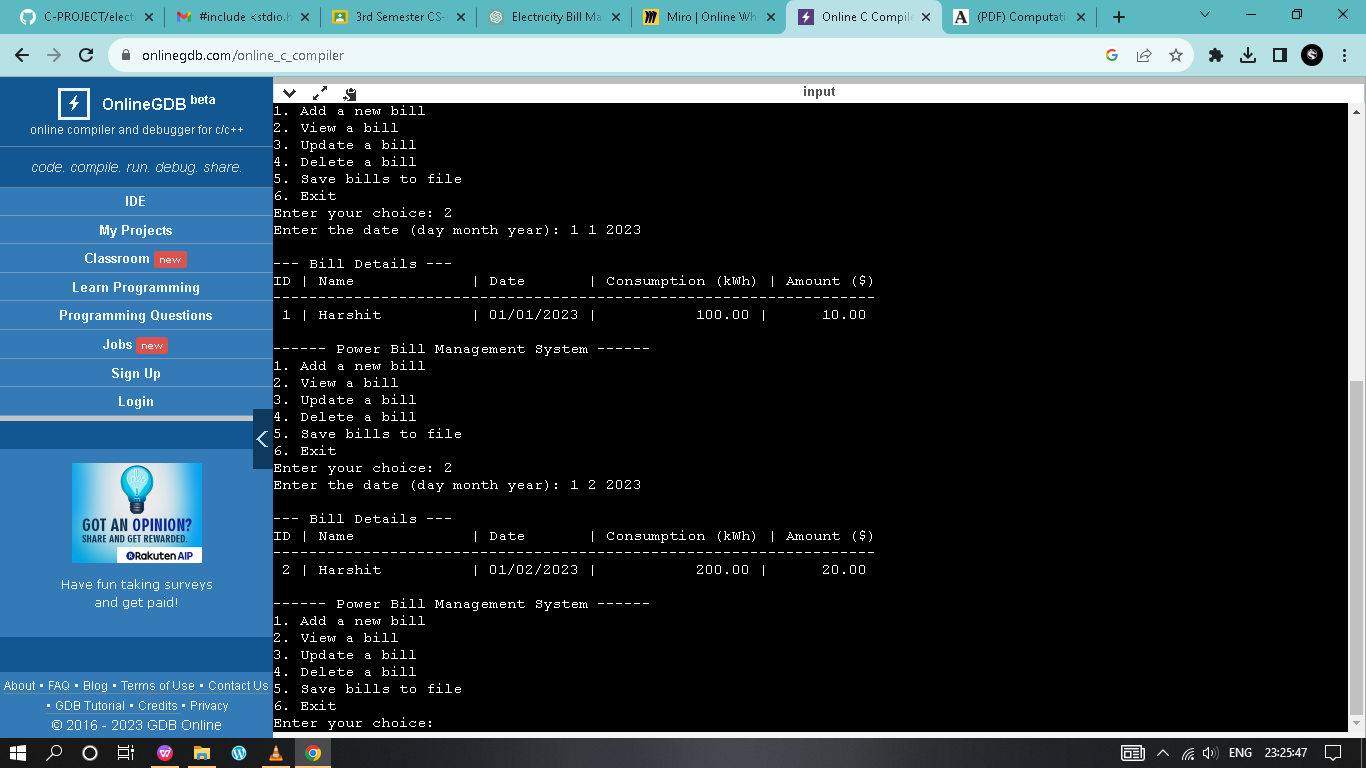
In conclusion, the project successfully automates the computation of transmission parameters in power systems, providing users with a convenient and efficient tool for power system analysis. The system's front-end design ensures user-friendly data input, while the back-end performs accurate calculations. The use of "data.db" as a database allows for data storage and retrieval. Challenges were addressed, and the project holds potential for future enhancements.

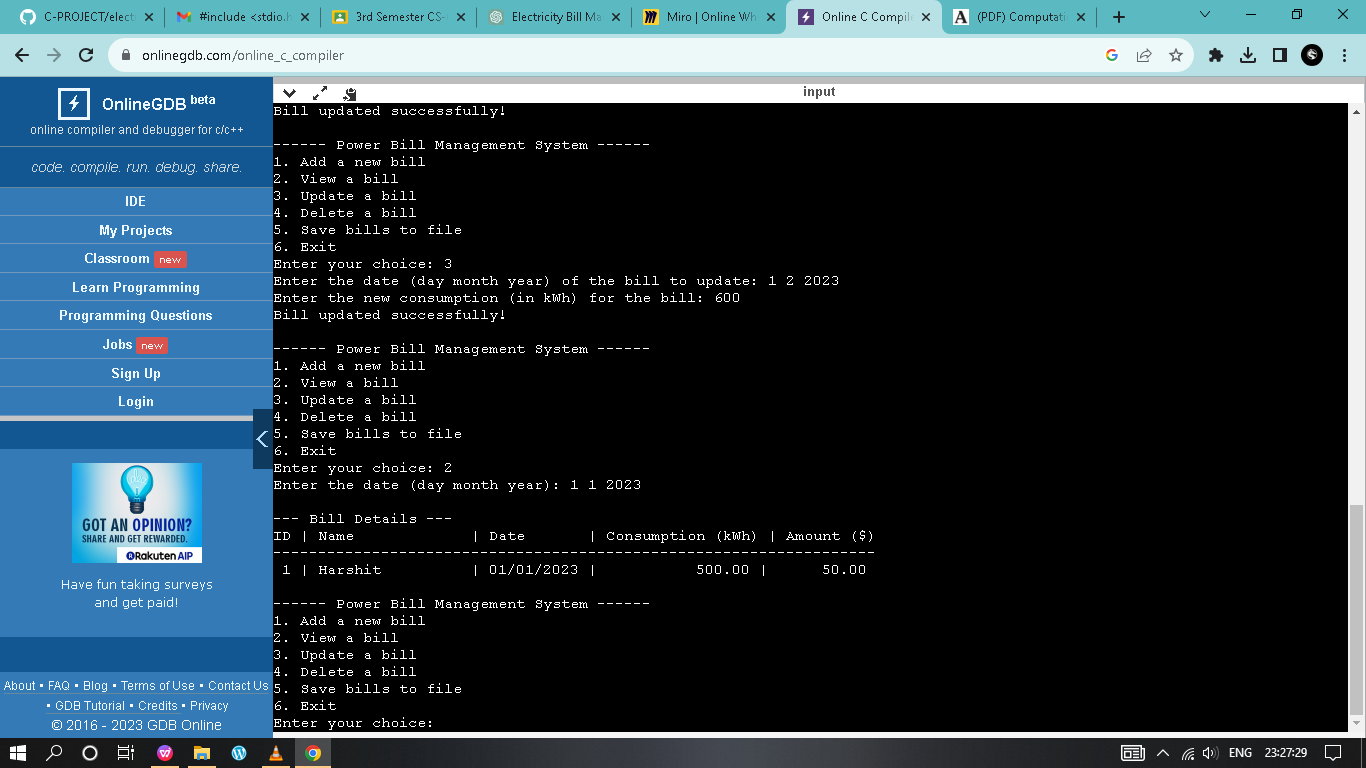
**12. References**

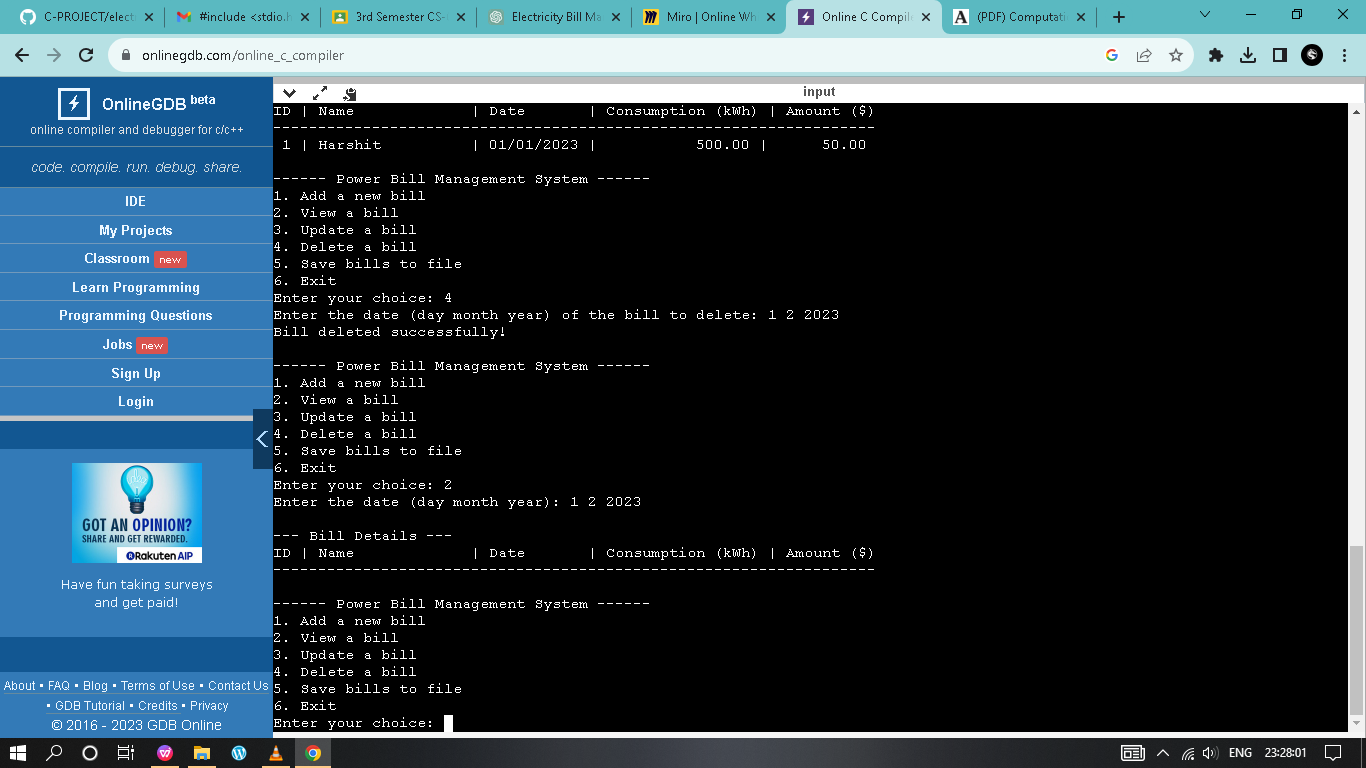
1. <https://miro.com/app/dashboard/>
2. <https://www.onlinegdb.com/online_c_compiler>
3. [www.academia.edu](http://www.academia.edu)
4. **Appendices**
   1. **Screenshots**

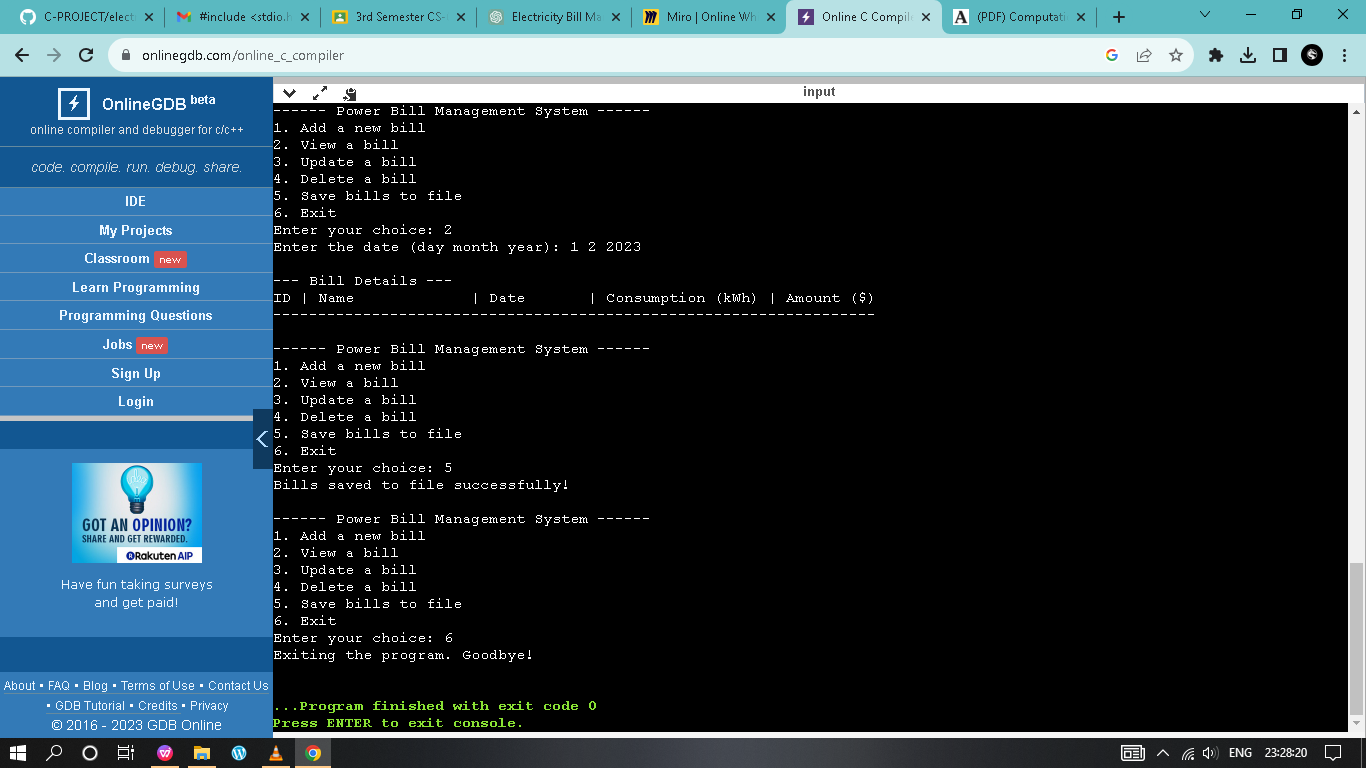


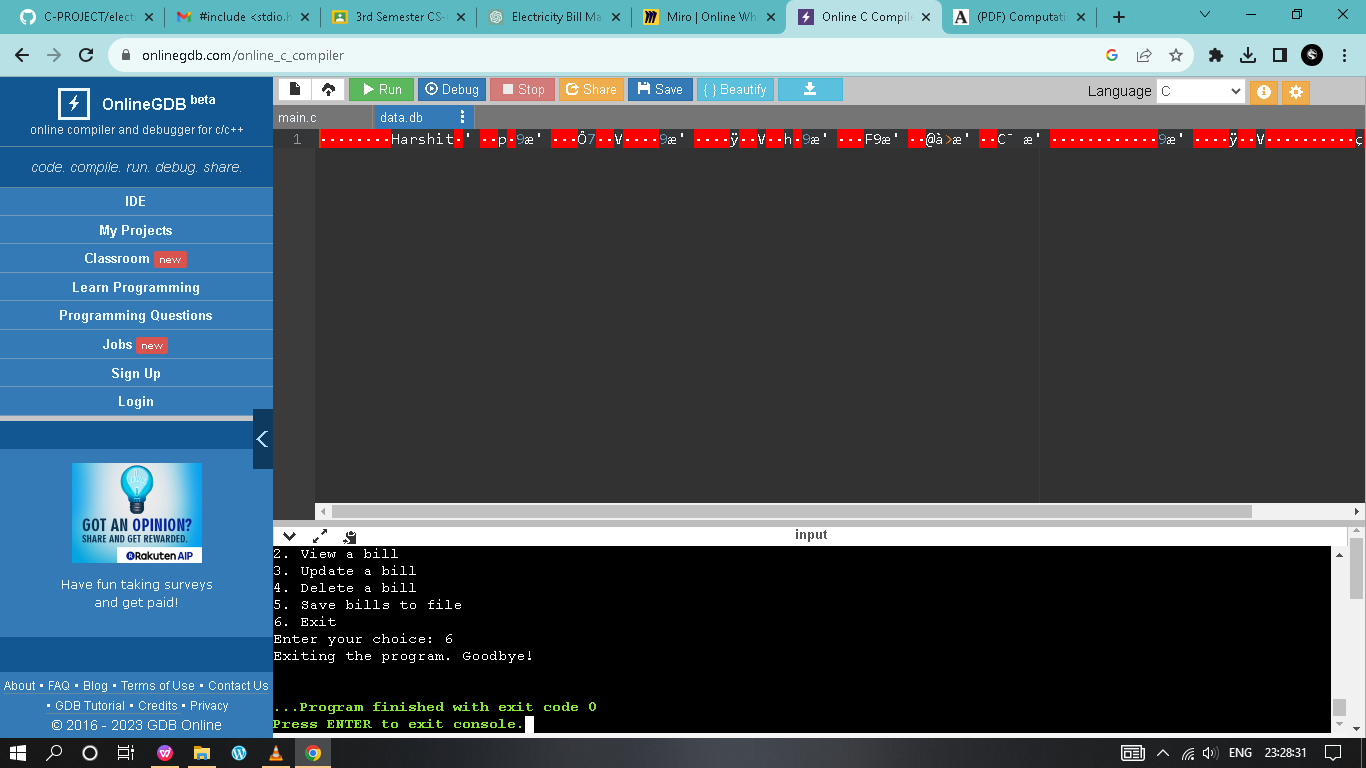












* 1. **Code Snippets**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**// Define the maximum number of bills the system can handle**

**#define MAX\_BILLS 100**

**// Structure to represent a power bill**

**struct PowerBill**

**{**

**int id;**

**char name[100];**

**int day;**

**int month;**

**int year;**

**float consumption;**

**float amount;**

**};**

**// Function prototypes**

**int addBill (struct PowerBill bills[], int \*count);**

**void viewBills (struct PowerBill bills[], int count);**

**void updateBill (struct PowerBill bills[], int count);**

**void deleteBill (struct PowerBill bills[], int \*count);**

**void saveBillsToFile (struct PowerBill bills[], int count);**

**void loadBillsFromFile (struct PowerBill bills[], int \*count);**

**int**

**main ()**

**{**

**struct PowerBill bills[MAX\_BILLS];**

**int billCount = 0;**

**int choice;**

**// Load bill data from file at program start**

**loadBillsFromFile (bills, &billCount);**

**do**

**{**

**// Display the menu**

**printf ("\n------ Power Bill Management System ------\n");**

**printf ("1. Add a new bill\n");**

**printf ("2. View a bill\n");**

**printf ("3. Update a bill\n");**

**printf ("4. Delete a bill\n");**

**printf ("5. Save bills to file\n");**

**printf ("6. Exit\n");**

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

**if (scanf ("%d", &choice) != 1)**

**{**

**printf ("Invalid input. Please enter a valid number.\n");**

**while (getchar () != '\n'); // Clear the input buffer**

**continue;**

**}**

**switch (choice)**

**{**

**case 1:**

**billCount = addBill (bills, &billCount);**

**break;**

**case 2:**

**viewBills (bills, billCount);**

**break;**

**case 3:**

**updateBill (bills, billCount);**

**break;**

**case 4:**

**deleteBill (bills, &billCount);**

**break;**

**case 5:**

**saveBillsToFile (bills, billCount);**

**break;**

**case 6:**

**printf ("Exiting the program. Goodbye!\n");**

**break;**

**default:**

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

**}**

**}**

**while (choice != 6);**

**return 0;**

**}**

**int**

**addBill (struct PowerBill bills[], int \*count)**

**{**

**if (\*count >= MAX\_BILLS)**

**{**

**printf ("Cannot add more bills. The system is full.\n");**

**return \*count;**

**}**

**// Assign a systematically generated ID**

**int newId = \*count + 1;**

**struct PowerBill newBill;**

**newBill.id = newId;**

**printf ("Enter the user's name: ");**

**if (scanf ("%99s", newBill.name) != 1)**

**{**

**printf ("Invalid input for name. Please enter a valid name.\n");**

**while (getchar () != '\n'); // Clear the input buffer**

**return \*count;**

**}**

**printf ("Enter the date (day month year): ");**

**if (scanf ("%d %d %d", &newBill.day, &newBill.month, &newBill.year) != 3)**

**{**

**printf**

**("Invalid date format. Please use numbers for day, month, and year.\n");**

**while (getchar () != '\n'); // Clear the input buffer**

**return \*count;**

**}**

**printf ("Enter the consumption (in kWh): ");**

**if (scanf ("%f", &newBill.consumption) != 1)**

**{**

**printf**

**("Invalid input for consumption. Please enter a valid number.\n");**

**while (getchar () != '\n'); // Clear the input buffer**

**return \*count;**

**}**

**// Calculate the amount**

**newBill.amount = newBill.consumption \* 0.10; // Assuming a fixed rate**

**bills[\*count] = newBill;**

**(\*count)++;**

**printf ("Bill added successfully!\n");**

**return \*count;**

**}**

**void**

**viewBills (struct PowerBill bills[], int count)**

**{**

**if (count == 0)**

**{**

**printf ("No bills found.\n");**

**return;**

**}**

**int day, month, year;**

**printf ("Enter the date (day month year): ");**

**if (scanf ("%d %d %d", &day, &month, &year) != 3)**

**{**

**printf**

**("Invalid date format. Please use numbers for day, month, and year.\n");**

**while (getchar () != '\n'); // Clear the input buffer**

**return;**

**}**

**printf ("\n--- Bill Details ---\n");**

**printf**

**("ID | Name | Date | Consumption (kWh) | Amount ($)\n");**

**printf**

**("-------------------------------------------------------------------\n");**

**for (int i = 0; i < count; i++)**

**{**

**if (bills[i].day == day && bills[i].month == month**

**&& bills[i].year == year)**

**{**

**printf ("%2d | %-16s | %02d/%02d/%d | %16.2f | %10.2f\n",**

**bills[i].id, bills[i].name, bills[i].day, bills[i].month,**

**bills[i].year,**

**bills[i].consumption, bills[i].amount);**

**}**

**}**

**}**

**void**

**updateBill (struct PowerBill bills[], int count)**

**{**

**if (count == 0)**

**{**

**printf ("No bills found. Nothing to update.\n");**

**return;**

**}**

**int day, month, year;**

**printf ("Enter the date (day month year) of the bill to update: ");**

**if (scanf ("%d %d %d", &day, &month, &year) != 3)**

**{**

**printf**

**("Invalid date format. Please use numbers for day, month, and year.\n");**

**while (getchar () != '\n'); // Clear the input buffer**

**return;**

**}**

**int found = 0;**

**for (int i = 0; i < count; i++)**

**{**

**if (bills[i].day == day && bills[i].month == month**

**&& bills[i].year == year)**

**{**

**printf ("Enter the new consumption (in kWh) for the bill: ");**

**if (scanf ("%f", &bills[i].consumption) != 1)**

**{**

**printf**

**("Invalid input for consumption. Please enter a valid number.\n");**

**while (getchar () != '\n'); // Clear the input buffer**

**return;**

**}**

**// Recalculate the amount**

**bills[i].amount = bills[i].consumption \* 0.10; // Assuming a fixed rate**

**printf ("Bill updated successfully!\n");**

**found = 1;**

**break;**

**}**

**}**

**if (!found)**

**{**

**printf ("No bill found for the specified date.\n");**

**}**

**}**

**void**

**deleteBill (struct PowerBill bills[], int \*count)**

**{**

**if (\*count == 0)**

**{**

**printf ("No bills found. Nothing to delete.\n");**

**return;**

**}**

**int day, month, year;**

**printf ("Enter the date (day month year) of the bill to delete: ");**

**if (scanf ("%d %d %d", &day, &month, &year) != 3)**

**{**

**printf**

**("Invalid date format. Please use numbers for day, month, and year.\n");**

**while (getchar () != '\n'); // Clear the input buffer**

**return;**

**}**

**int found = 0;**

**for (int i = 0; i < \*count; i++)**

**{**

**if (bills[i].day == day && bills[i].month == month**

**&& bills[i].year == year)**

**{**

**for (int j = i; j < (\*count) - 1; j++)**

**{**

**bills[j] = bills[j + 1];**

**}**

**(\*count)--;**

**printf ("Bill deleted successfully!\n");**

**found = 1;**

**break;**

**}**

**}**

**if (!found)**

**{**

**printf ("No bill found for the specified date.\n");**

**}**

**}**

**void**

**saveBillsToFile (struct PowerBill bills[], int count)**

**{**

**FILE \* file = fopen ("data.db", "wb");**

**if (file == NULL)**

**{**

**printf ("Error opening the file for writing.\n");**

**return;**

**}**

**fwrite (&count, sizeof (int), 1, file);**

**fwrite (bills, sizeof (struct PowerBill), count, file);**

**fclose (file);**

**printf ("Bills saved to file successfully!\n");**

**}**

**void**

**loadBillsFromFile (struct PowerBill bills[], int \*count)**

**{**

**FILE \* file = fopen ("data.db", "rb");**

**if (file == NULL)**

**{**

**printf ("No existing data file found.\n");**

**return;**

**}**

**fread (count, sizeof (int), 1, file);**

**fread (bills, sizeof (struct PowerBill), \*count, file);**

**fclose (file);**

**printf ("Bills loaded from file successfully!\n");**

**}**