IIT BHUBANESWAR

BANK MANAGEMENT SYSTEM

PROJECT BY

ABHIMANY SINGH (23EE01002)
ADITYA KUMAR GIRI (23ME01001)
ROHINISH RAJ (23EC01038)
HIMANSHU JAIN (23EE01019)

UNDER SUPERVISION OF

DR. DEVASHREE TRIPATHI

GitHub link:
https://github.com/Abhimanyu5100/PDS_Capstone_Project.git

CONTENT

INTRODUCTION

We have developed a bank management system where a user can create an account and then login using a unique login ID provided to him. We have named our bank “PLUTUS FINCORP” as ‘Plutus’ is the Greek goddess of wealth. The program is completely based on C programming language and runs solely on terminal. There are various features that our bank provides which we will be explaining later.

MOTIVATION

In today's digital age, the banking industry is continuously evolving, and efficient management systems are essential. Developing a bank management system project gave us the opportunity to contribute to this dynamic sector and gain practical experience.

Creating a bank management system involves addressing real-world challenges faced by banks, such as security, data management, and customer service. By working on this project, we were able to experience the functioning of banks in various aspects.

Building a bank management system required to learn various technologies, such as database management, security protocols, and software development. This project is a good and simplified version of a bank where we are providing different functionalities for the user.

The banking sector is constantly seeking innovations to enhance services, streamline operations, and improve customer experiences. By modification of our project, it could be an innovative solution that can potentially disrupt and improve the industry.

By working on this project, we also got to know many functions of C language that made our project a bit better.

We wanted to make a proper management system where we can handle data efficiently given by the user, so we found the bank management system as a appropriate project for us.

As we delve into the intricacies of bank management, we became more aware of the challenges banks face and the importance of sound financial practices. This knowledge can benefit us personally and professionally.

Working on a bank management system project involves collaboration with other individuals, including programmers, analysts, and domain experts. This experience helped us develop teamwork and communication skills.

By successfully completing this project of this magnitude is not just professionally satisfying but it also boosts our self-confidence and personal satisfaction.

OVERVIEW

This is a project which demonstrates the working of a bank on a very basic level for a people to understand. When a person enters the bank, they are presented with a welcome screen that prompts them either register as a new user or login as an existing user. After registration / login, we present to the user various features of the bank such as Loans, Fixed Deposits (FD) and Insurance.

If the user opts for any of our services, then they will be presented the service-specific options. Additionally, there is an option to exit the bank if the user wishes to leave by choosing the exit option.

DEVELOPMENT PROCESS

We decided on the topic and then did some research about our project. There were so many concepts that were new to us when the project was assigned to us, so we started learning about the topics from YouTube and books. After we all felt confident about our respective parts, we started writing our code. We had two more ideas which we started working on but the results were not satisfactory. This code also required so many efforts as there were so many errors. This code contains functions for all working mechanisms in our bank.

ABSTRACT

The Bank Management System is a comprehensive software application designed to streamline and automate various banking operations, including user registration, Fixed Deposit (FD) management, deposit and withdrawal transactions, insurance management, and loan processing. This project aims to provide an efficient and user-friendly platform for both bank employees and customers, enhancing the overall banking experience.

1. **User Registration**:

   The system offers a user-friendly registration process that allows customers to create their accounts, providing personal information, contact details, and other necessary details securely. It ensures data privacy and complies with regulatory requirements. Bank employees can also use this feature to create and manage customer accounts.

2. **Fixed Deposit (FD) Management:**

   Customers can open Fixed Deposit accounts, specifying the amount, tenure, and interest rate. The system calculates and displays the maturity amount and interest accrued. The system also calculates pre maturity amount for User Who wants to know.

3. **Deposit and Withdrawal Transactions**:

   The system enables customers to perform various banking transactions, including deposits and withdrawals. Customers can deposit funds into their savings or current accounts and withdraw money as needed. Bank employees can process these transactions, ensuring secure and accurate financial operations.

4. **Insurance Management:**

Customers can explore various insurance policies offered by the bank, compare them, and choose the one that suits their needs. The system provides three insurance policies i.e.

- Life Insurance
- Health Insurance
- Car Insurance

5. **Loan Processing:**

Customers can apply for loans using the system, providing the necessary information and supporting documents. Bank employees can review loan applications, check eligibility, and approve or reject them based on predefined criteria. The system calculates EMI (Equated Monthly Instalments) and provides detailed loan information to the customer. The system provides given loan on the basis of User CIBIL Score:

- Home Loan
- Business Loan
- Student Loan
- Personal Loan

FUNCTION DESCRIPTION

1. REGISTRATION BLOCK:

- **void show_Loading():**
  This function is designed to display a simple console-based loading animation using characters from the char array. It does this by printing a message with a rotating character and updating it in a loop with a small delay between iterations.

- **int registration_driver():**
  This code is part of a banking application and provides registration and login functionalities. Here's a concise summary of its key actions:

  - It collects user information for registration, including name, age, phone number, and password.
  - It generates a random account number and stores the user's data in a CSV file.
  - For login, it checks user credentials against the CSV file and grants access if successful.
  - The code handles file operations and provides error checking.

2. FD BLOCK :

- **float CalculateInterestRate():**

This function calculates the interest rate for a fixed deposit based on the deposit duration and whether the account holder is a senior citizen. Here's a simplified explanation:

- The function takes two parameters: duration (the deposit duration in months) and isSeniorCitizen (a flag for senior citizen status).
- It uses conditional statements to determine the interest rate.
- If the account holder is a senior citizen, it offers higher interest rates compared to non-senior citizens.
- The function defines interest rate intervals for different deposit durations and returns the corresponding rate.
- This function is typically used in financial applications, such as fixed deposit interest rate calculators, to provide tailored interest rates based on customer characteristics and deposit duration.
-

- **float CalculatePreMaturityInterest():**
  This function calculates pre-maturity interest rates for fixed deposits based on the deposit duration and the account holder's senior citizen status. It provides higher rates for senior citizens and varies the rates according to the deposit duration. This function is commonly used in financial applications to calculate interest payments for fixed deposits.

- **void loading_screen():**
  This function is designed to display a simple console-based loading animation using characters from the char array. It does this by printing a message with a rotating character and updating it in a loop with a small delay between iterations.

- **Void FD():**
  The FD function is part of a financial application for managing fixed deposit (FD) accounts. Here's a concise overview of its main actions:

  - It initializes variables and opens a file to store account data.
  - It collects user input for account details, including name, account number, deposit amount, duration, and senior citizen status.
  - It displays loading screens for user feedback.
  - It calculates and displays expected percentage return rates for the FD based on senior citizen status.
  - It provides options to calculate expected returns before and after maturity.
  - It generates a random FD account number.
- **int fd_driver():**

This function is a driver function that calls the FD function, which appears to be responsible for managing fixed deposit (FD) accounts. The actual fd_driver function is defined later in the code. It calls the FD function, which manages FD accounts, and then returns an integer value of 0.

3. INSURANCE BLOCK:

- **int policyNum() :**
  The policyNum function generates a random policy number within a specific range and returns it. Here's a concise explanation of what this function does:
  - It uses the rand() function to generate a random integer.
  - The generated random integer is constrained to a range using the modulo operator (%) and addition. In this case, it creates a random number between 1,000,000,000 and 9,999,999,999.
  - The function assigns this random number to a variable named policyNum.
  - Finally, it returns the generated policy number as the result of the function.

- **void Show_load():**
  This function is designed to display a simple console-based loading animation using characters from the char array. It does this by printing a message with a rotating character and updating it in a loop with a small delay between iterations.

- **int insurance():**
  The insurance function is designed for managing different types of insurance policies, including life, health, and car insurance. Here's a concise overview of its primary actions:

  - It opens separate files for storing data related to different insurance policies (life, health, car).
  - It initializes various variables and handles file opening errors.
  - It presents a menu for the user to choose between different insurance types.

  1. Life Insurance:

     - When the user selects "Life Insurance," it collects policy holder and beneficiary information.
     - It allows the user to choose the coverage amount (e.g., 50 lakhs, 1 crore, or 2 crores) and calculates the associated monthly premium.

- It provides options for the user to continue with the insurance application or cancel it.
- If the user chooses to continue, it displays the details of the insurance policy, including policy holder name, beneficiary, coverage amount, policy number, and monthly premium.
- It saves the insurance policy details to a file and closes the file.

2. Health Insurance:

   - After choosing "Health Insurance" from the menu, the user enters policy holder and beneficiary information.
   - The user has to declare whether he smokes or not.
   - The user selects the coverage amount (e.g., 2 lakhs, 5 lakhs, 10 lakhs or 20 lakhs), and the function calculates the monthly premium based on the chosen coverage.
   - The user is given the option to continue with the insurance application or cancel it.
   - If the user chooses to continue, the function displays the insurance policy details, including the policy holder's name, beneficiary, coverage amount, policy number, and monthly premium.
   - The function saves the health insurance policy details to a file and closes the file.

3. Car Insurance:
   - Upon choosing "Car Insurance," the user provides their name, age, price of car and the age of their car.
   - The function calculates the premium based on the car's age.
   - The user is given the option to continue with the car insurance application or cancel it.
   - If the user chooses to continue, the function displays the car insurance policy details, including the policy holder's name, car age, coverage type, policy number, and monthly premium.
   - The function saves the car insurance policy details to a file and closes the file.

- **int insurance_driver() :**

  This function is a driver function that calls the Insurance function, which appears to be responsible for Insurance policies. The actual insurance_driver function is defined later in the code. It calls the insurance function, which manages all Insurance policies.

4. LOAN BLOCK:

- **void home_loan():**
  This function is responsible for guiding a user through the process of obtaining a home loan. It takes the user's credit score as input and proceeds as follows:
  - It displays the maximum loan amount the user can get based on their credit score.
  - It prompts the user to enter the desired loan amount in lakhs and validates that it's within an acceptable range.
  - It then prompts the user to enter the loan tenure in months and ensures it falls within the allowed tenure range.
  - The user is asked to input their monthly income in rupees.
  - The function likely calculates the interest rate, approved loan amount, and monthly EMI based on the user's inputs (credit score, loan amount, tenure, and income).
  - It displays the calculated loan details, including the approved loan amount, tenure, annual interest rate, and monthly EMI to the user.
  - It gives the user the option to either accept the loan or exit the program.
    - If the user accepts the loan, it likely calls another function (final_output) to display the final loan details.
    - If the user chooses to exit, the function ends.

- **void business_loan():**
  This function helps a user evaluate and apply for a home loan. The function takes the user's credit score as input. Here's how it works:
  - It provides information to the user about the maximum loan amount they can get based on their credit score.
  - The user is asked to input the desired loan amount in lakhs, and the code ensures that the amount is within acceptable limits.
  - Next, the user is prompted to enter the loan tenure in months, and the code checks that it falls within a valid tenure range.
  - The user is also asked to input their monthly income in rupees.
  - The function likely calculates the interest rate, approved loan amount, and monthly EMI based on the user's inputs, including their credit score, desired loan amount, tenure, and income.
  - The calculated loan details, including the approved loan amount, loan tenure, annual interest rate, and monthly EMI, are displayed to the user.
  - The user is given the choice to either accept the loan or exit the program:
    - If the user accepts the loan, it likely calls another function (final_output) to provide the user with the final loan details.
    - If the user chooses to exit, the function concludes.

- **void student_loan() :**

  This function is designed to assist a user in evaluating and applying for a student loan. It takes the user's credit score as input and follows this process:
  - It informs the user about the maximum loan amount they can potentially receive based on their credit score.
  - The user is asked to input their desired loan amount in lakhs, and the code ensures that this amount falls within acceptable limits.
  - Next, the user is prompted to specify the loan tenure in months, with the code checking that the tenure is within a valid range.
  - The user is also asked to provide their monthly income in rupees.
  - The function likely calculates the interest rate, approved loan amount, and monthly EMI based on the user's inputs, including their credit score, desired loan amount, tenure, and income.
  - The calculated loan details, including the approved loan amount, loan tenure, annual interest rate, and monthly EMI, are displayed to the user.
  - The user is given the choice to either accept the loan or exit the program:
    - If the user accepts the loan, it likely calls another function (final_output) to provide the user with the final loan details.
    - If the user chooses to exit, the function concludes

- **void personal_loan() :**

  The personal_loan function is designed to help a user evaluate and apply for a personal loan. It takes the user's credit score as input and follows this process:
  - It informs the user about the maximum loan amount they can potentially receive based on their credit score (max_loan).
  - The user is asked to input their desired loan amount in lakhs, and the code ensures that this amount falls within acceptable limits.
  - Next, the user is prompted to specify the loan tenure in months, with the code checking that the tenure is within a valid range.
  - The user is also asked to provide their monthly income in rupees.
  - The function likely calculates the interest rate, approved loan amount, and monthly EMI based on the user's inputs, including their credit score, desired loan amount, tenure, and income.
  - The calculated loan details, including the approved loan amount, loan tenure, annual interest rate, and monthly EMI, are displayed to the user.
  - The user is given the choice to either accept the loan or exit the program:
    - If the user accepts the loan, it likely calls another function (final_output) to provide the user with the final loan details.

- If the user chooses to exit, the function concludes.

- **void loan_amount_tracker() :**
  This function calculates the maximum loan amount a user can obtain based on their credit score and sets the value of 'max_loan' accordingly. This information can be used in subsequent loan application processes to inform the user about the loan amount they are eligible for.
- **void interest_home() :**
  This function determines the interest rate for a home loan based on the user's credit score. It then calculates the Equated Monthly Instalment (EMI) for the loan. Depending on the user's monthly income and EMI, it approves a loan amount. The function covers various income and EMI scenarios to provide a suitable loan offer to the user.

- **void interest_student() :**
  This function determines the interest rate for a student loan based on the user's credit score. It then calculates the Equated Monthly Instalment (EMI) for the loan. Depending on the user's monthly income and EMI, it approves a loan amount. The function covers various income and EMI scenarios to provide a suitable loan offer to the user.

- **void interest_personal() :**

  This function determines the interest rate for a personal loan based on the user's credit score. It then calculates the Equated Monthly Instalment (EMI) for the loan. Depending on the user's monthly income and EMI, it approves a loan amount. The function covers various income and EMI scenarios to provide a suitable loan offer to the user.

- **void interest_buisness() :**
  This function determines the interest rate for a buisness loan based on the user's credit score. It then calculates the Equated Monthly Instalment (EMI) for the loan. Depending on the user's monthly income and EMI, it approves a loan amount. The function covers various income and EMI scenarios to provide a suitable loan offer to the user.

- **void final_output() :**
  The function provides the user with the final loan approval details. It displays a congratulatory message and presents essential loan information, including the approved loan amount, loan tenure, interest rate, and EMI. This function serves as the endpoint of the loan application process, informing the user about the loan's terms and conditions.

- **void show_loading() / void show_processing2()/void show_wait() :**
  This function is designed to display a simple console-based loading animation using characters from the char array. It does this by printing a message with a rotating character and updating it in a loop with a small delay between iterations.

TESTING AND DEBUGGING

We were facing a large number of errors in our initial runs. This was expected as we were using new concepts. Solving these errors provided us significant insight of the code which was necessary for understanding the basic underlying concepts. For debugging, we introduced various print statements to understand the flow of the code as well as tackled any new error with enthusiasm. Traversing the struct and comparing the value was a major contributor to the errors. These were solved after using the right format specifier as well as logic.

CONTRIBUTION

The project work is broadly divided among four of our team members. Talking about the division of code block, the login window is prepared by Abhimanyu. Main block and Loan section is prepared by Himanshu. The FD section is prepared by Rohinish and the Insurance section is prepared by Aditya. Adding colours to the text was done by Himanshu. The idea of preparing a loading animation is put by Abhimanyu. Aditya and Rohinish mostly focused on testing and debugging of the final code finding out errors and fixing them. Preparation of report is largely done by Aditya, Rohinish and Abhimanyu.

WHAT WE LEARNT

1. File handling is a crucial part that we learnt.
2. Using functions such as srand and rand to generate pseudo-random number.
3. Using Github for collaborative working.
4. Exploring new libraries such as <time.h> .
5. Use of ChatGPT as a helpful tool.

AREAS OF IMPROVEMENT

The improvements that we would have made if we were provided more time and experience is working on Data Encryption, Enhanced system Checks, Graphical user Interface (GUI) and Accessibility. Let's break down each point:

1. Data Encryption:
    - Encrypting personal information is a crucial step for confidentiality of user data. Technologies like SSL/TLS communication encryption algorithms for data.

2. Enhanced System checks:
    - Implementing through error handling and validation mechanism is essential for identifying potential issues promptly.
    - There are some parts where user need to input a number to proceed or exit, if user inputs the correct number the code goes on smoothly. If user enters a wrong number the output is invalid input but when the input is anything except integer the terminal crashes.

3. Graphical User Interface:
    - A graphical user interface can significantly improve the user friendliness of system.
    - GUIs make it easier for user to interact with the system and significantly reduce the learning curve that is required to access an application.

4. Accessibility:
    - Consideration of user with disability. E.g., incorporating a screen reader.

5. More areas
    - In file handling the data is getting stored only for the registration section, if we have got more time and experience then we could have tried to solve this problem and added a separate file to print the data the user has given previously.

CONCLUSION

Overall, it was a worthwhile experience developing a bank management system. We learned many things from this project, we also learned how we can store data from previous iterations and then use it to jump right back to where we left off when the code run again, also not just coding knowledge but also thinking process of developing a product as well as how we can tackle errors, and how deriving a systematic procedure can significantly increase productivity and efficiency. Overall, this experience will surely help in our future endeavour in the field of programming and data structures.