Budget Tracker

## *Introduction*

Welcome to the Budget Tracker User Guide, your go-to resource for effectively managing your personal finances. In an era where financial awareness and control are paramount, our budget tracker application is designed to empower you with the tools needed to take charge of your money.

This comprehensive guide will walk you through every aspect of our budget tracker, from creating your account and logging in to adding income, tracking expenses, and gaining insights into your financial health. We believe that managing your finances should be accessible, intuitive, and, most importantly, effective. With our budget tracker, you'll be able to achieve just that.

## Getting Started: Your Financial Journey Begins Here

**Creating an Account:**

* Your path to financial freedom commences with the creation of your personal user account. This account is not merely a gateway to our budget tracker; it's your secure vault where your financial data is meticulously safeguarded. Simply select a unique username and password to establish your financial profile.

**Logging In:**

* After your account is created, accessing your financial world is as easy as entering your chosen username and password. Our robust security measures ensure that your sensitive financial information remains confidential and protected.

*Managing Your Finances: Tailoring Financial Solutions to Your Needs*

**Adding Income:**

* Our budget tracker has been meticulously crafted to simplify the process of income management. Forget about the complexities of income categorization. Instead, effortlessly input your income amount and describe its source. Feel free to add multiple income sources simultaneously, streamlining the process.

**Adding Expenses:**

* Similar to income, the process of adding expenses is designed to be seamless and user-friendly. Enter each expense individually, specifying the amount, providing a brief description, and selecting from predefined categories such as House, University, and Food. Whether you have a single expense or a multitude to record, our program adapts effortlessly to your needs.

**Expense Categories:**

* Our commitment to making financial management accessible extends to the inclusion of expense categories. These categories offer a robust organizational framework, simplifying the task of tracking and understanding your spending habits. Whether you're allocating expenses to "Household," "Education," or "Food," our budget tracker allows you to take control of every aspect of your financial life.

*Monitoring Your Finances: Real-Time Insights for Informed Decisions*

**Calculating Current Balance:**

* At any juncture, you can swiftly calculate your current financial balance by selecting this option. The program instantaneously processes your income and expense data, providing you with a real-time snapshot of your financial health. Armed with this knowledge, you can make informed decisions about your spending and savings.

**Viewing Income and Expense History:**

* The ability to review your financial history is a cornerstone feature of our budget tracker. Access your comprehensive income and expense history, complete with detailed entries, amounts, and descriptions. Whether you're analyzing your financial past or planning for the future, this tool offers invaluable insights.

*Setting Financial Goals: Envisioning Your Financial Future*

**Savings Goal:**

* As part of our commitment to continuous improvement, we are actively working on incorporating a savings goal feature. This forthcoming addition will empower you to set specific savings objectives and monitor your progress toward realizing your financial dreams.

*Conclusion: Your Journey to Financial Empowerment Begins Now*

Our budget tracker is a powerful and user-friendly tool that puts you in command of your financial destiny. With an intuitive interface, robust security, and flexible income and expense management, this application is your key to unlocking the full potential of your finances.

Whether you're striving to achieve savings goals, understand your spending habits, or simply maintain a closer eye on your financial well-being, our budget tracker is your steadfast companion on this journey. We appreciate your trust in our budget tracker, and we are here to support you every step of the way.

Begin your path to financial empowerment today with our Budget Tracker!

## *The importance of Financial Management: Achieving your goals*

In an ever-evolving and interconnected world, the significance of effective financial management transcends mere fiscal responsibility; it shapes the very trajectory of your life. Whether you aspire to own a home, embark on exciting adventures, invest in your education, or savor a comfortable retirement, prudent financial management stands as the compass guiding you toward your goals.

*1. Building a Secure Future:*

Financial management isn't solely about budgeting or saving for a rainy day; it's about forging a secure future for yourself and your loved ones. When you allocate your resources wisely, you create a financial safety net that shields you from unforeseen challenges, such as medical emergencies or unexpected job loss. This financial security is the foundation upon which you can confidently build your dreams.

*2. Owning a Home:*

For many, owning a home is the epitome of the American Dream. Effective financial management helps you embark on the journey to homeownership by enabling you to save for a down payment, maintain a good credit score, and budget for mortgage payments and associated costs. It transforms what may seem like an unattainable dream into a concrete goal within reach.

*3. Education and Career Advancement:*

Education is a potent instrument of personal and professional growth. Whether you're seeking a college degree, vocational training, or advanced certifications, financial management empowers you to invest in your education. It enables you to set aside funds for tuition, purchase essential study materials, and even consider career advancement opportunities that may require further training or qualifications. Your ability to manage finances effectively can make education not just accessible but also an avenue for realizing your career aspirations.

*4. Financial Independence:*

Financial management is your ticket to freedom and independence. When you master the art of budgeting, saving, and investing, you liberate yourself from financial constraints. No longer bound by financial stress or debt, you have the autonomy to make choices that align with your values and desires, be it starting a business, pursuing creative endeavors, or exploring new horizons.

*5. Comfortable Retirement:*

A comfortable retirement is a goal cherished by many. Effective financial management is the compass that guides you on the path to retirement security. By diligently saving and investing throughout your working years, you pave the way for a retirement filled with relaxation, travel, and the pursuit of lifelong passions. It ensures that your golden years are truly golden.

*6. Weathering Economic Challenges:*

The world is no stranger to economic uncertainties, from market downturns to global crises. Effective financial management equips you with the resilience to weather these challenges. With a well-prepared financial plan, you can mitigate the impact of economic downturns, safeguard your investments, and adapt to changing circumstances without undue stress.

*7. Accomplishing Life's Adventures:*

Life is an adventure filled with exciting opportunities and unforeseen surprises. Financial management provides you with the financial freedom to seize these moments. Whether it's embarking on a world tour, pursuing a passion project, or supporting a charitable cause close to your heart, prudent financial management ensures that you have the means to pursue your passions and dreams.

In conclusion, the importance of financial management cannot be overstated. It is the cornerstone upon which your life goals and dreams are built. By mastering the art of financial management, you gain control over your financial destiny, ensuring that you are well-prepared to achieve your aspirations and navigate life's challenges with confidence and resilience.

## *The Role of Technology: A Revolution in Personal Finance*

In the ever-evolving landscape of personal finance, technology stands as a transformative force, reshaping the way individuals manage their money and make financial decisions. The emergence of digital tools and platforms has democratized financial management, making it accessible to people of all backgrounds and levels of expertise.

*1. Accessibility and Convenience:*

Technology has dismantled the barriers that once limited access to financial information and services. With the proliferation of smartphones and internet connectivity, individuals now have a wealth of financial resources at their fingertips. This accessibility empowers people to monitor their finances, conduct transactions, and gain financial insights with unparalleled convenience.

*2. Real-Time Financial Insights:*

One of the most remarkable advancements facilitated by technology is the ability to access real-time financial data. Gone are the days of waiting for monthly paper statements or manually recording transactions in ledgers. With the advent of online banking, mobile apps, and financial tracking tools like our budget tracker, individuals can instantly view their account balances, track income and expenses, and gauge their financial health with just a few taps or clicks.

*3. Automation and Efficiency:*

Technology has automated and streamlined many financial processes, saving time and reducing the risk of human error. Automated bill payments, for example, ensure that essential expenses are settled promptly, eliminating late fees and improving credit scores. Our budget tracker harnesses automation to simplify income and expense tracking, allowing users to effortlessly record and categorize financial transactions.

*4. Enhanced Security:*

In an era of heightened concern about data security, technology has responded with robust encryption and authentication measures. These security features safeguard sensitive financial information, ensuring that it remains confidential and protected from unauthorized access. Our budget tracker prioritizes the security of your financial data, implementing encryption protocols and secure login procedures to provide peace of mind.

*5. User-Friendly Interfaces:*

Perhaps one of the most significant benefits of technology in personal finance is the development of user-friendly interfaces. User experience (UX) design principles have been applied to financial applications, making them intuitive and accessible to users of all skill levels. Our budget tracker is a prime example of this, featuring a user-friendly interface that simplifies income and expense management, allowing you to focus on your financial goals rather than grappling with complex software.

*6. Data Analysis and Insights:*

Technology has enabled sophisticated data analysis and reporting, providing individuals with invaluable insights into their financial habits and trends. Our budget tracker employs data analytics to generate customized reports, helping you identify spending patterns, set savings goals, and make informed financial decisions.

In conclusion, the role of technology in personal finance cannot be overstated. It has democratized financial management, making it accessible, efficient, and secure. Our budget tracker leverages cutting-edge technology to deliver a seamless and user-friendly experience, empowering you to take control of your finances with confidence and ease. Whether you're tracking income, categorizing expenses, or setting savings goals, technology is your ally on the path to financial success.

## Chapter II: Algorithm

Algorithms are the backbone of computer science and, by extension, much of modern life. An algorithm is a step-by-step, well-defined procedure or a set of instructions for solving a specific problem or performing a particular task. These tasks can range from simple calculations to complex operations like sorting massive datasets, searching for information on the internet, or optimizing routes for package delivery.

Algorithms are everywhere, silently working behind the scenes in software applications, systems, and services we use daily. Whether it's the algorithm that suggests your next video on a streaming platform or the one that helps you find the quickest route to your destination using a navigation app, algorithms are the unsung heroes of the digital age.

Key characteristics of algorithms include efficiency, correctness, and reproducibility. An efficient algorithm finds a solution in the least amount of time and resources, while a correct algorithm produces accurate results. The ability to replicate the same results when given the same input is fundamental to the reliability of algorithms.

Algorithms play a pivotal role in solving problems across various domains, from mathematics and data analysis to artificial intelligence and machine learning. Understanding algorithms and their design is central to computer science and software development, as it empowers developers and engineers to create more efficient, reliable, and innovative solutions.

In the context of the provided C++ code for the budget tracker, we'll delve into the specific algorithms and procedures used to manage user data, input financial transactions, and provide financial insights. These algorithms are essential for creating a practical and user-friendly tool for personal finance management.

1. Header Files and Libraries:

* **Header Files**:
  + Header files in C++ are used to declare and define functions, classes, and objects that are utilized in the source code. They allow for modularization and organization of code by separating declarations from the implementation.
  + **Header files used in the program**:
    - **<iostream>**: This is the standard C++ input and output stream library. It's used for reading user input from the console and displaying output to the console. It's a fundamental part of most C++ programs.
    - **<fstream>**: The file stream library is used for file input and output operations. It allows the program to read from and write to files. In the budget tracker program, this is vital for saving and loading user data to and from files.
    - **<string>**: The string library provides string data types and various string manipulation functions. It's crucial for working with user-defined strings, such as usernames, descriptions, and categories.
    - **<vector>**: Vectors are part of the Standard Template Library (STL) and provide dynamic arrays. In the program, vectors are used to store lists of transactions, both income and expenses. They offer flexibility in handling a variable number of entries.
    - **<iomanip>**: This library is used for manipulating input and output formatting, especially for controlling the precision of floating-point numbers. It ensures that monetary values are displayed correctly with two decimal places.
* **Libraries**:
  + Libraries in C++ are collections of pre-compiled functions and code that can be used in a program. They extend the capabilities of C++ and offer a wide range of functionalities.
  + **Library used in the program**:
    - **Standard Library**: The program relies primarily on the C++ Standard Library, which includes the aforementioned header files as well as other standard libraries not explicitly mentioned. The C++ Standard Library provides a vast array of functions, data types, and containers, making it a versatile and essential tool for C++ programmers.

**Why These Choices**:

* **Modularity**: Header files allow for code modularity and organization. Declarations can be separated from the implementation, making the code more manageable and readable.
* **Functionality**: The selected header files and libraries cover essential functionalities required for this program, such as user input/output, file handling, string manipulation, and data storage.
* **Standardization**: The C++ Standard Library is widely supported and standardized, making it a reliable choice for a broad range of C++ applications.
* **Versatility**: These header files and libraries offer versatile tools for common programming tasks, which is particularly important in the context of the budget tracker application, where various functions and data types are required.

2. Data Structures:

* **Transaction Struct:**
  + *Purpose*: The **Transaction** struct plays a central role in representing financial transactions within the budget tracker application. It has three primary fields:
    - **description**: This field is a string that provides a brief, human-readable description of the transaction. For example, "Salary" for an income transaction or "Groceries" for an expense transaction.
    - **amount**: The **amount** field is a floating-point number that quantifies the financial value of the transaction. It represents the monetary amount associated with the transaction. For instance, "$100.0" might represent an income of $100, while "$30.5" could signify an expense of $30.50.
    - **category** (specific to expenses): In the case of expense transactions, the **category** field categorizes the expense into predefined or custom categories. For example, categories could include "Food," "Rent," "Taxes," "Bills," or the user can specify a custom category like "Entertainment."
  + *Why This Choice*: The choice of the **Transaction** struct is particularly advantageous for the following reasons:
    - **Data Encapsulation**: It encapsulates all the essential attributes of a financial transaction within a single data structure, making the code more organized and comprehensible.
    - **Simplicity**: It simplifies the management and manipulation of transaction data. Each transaction is represented as an instance of this struct, enabling straightforward access and modification.
    - **Flexibility**: The inclusion of a **category** field allows for flexible categorization of expense transactions, accommodating both predefined categories and user-defined categories.
    - **Readability and Clarity**: Using a struct helps improve code readability by creating a well-defined structure for transactions. It ensures that the description and amount are directly associated with each transaction.
* UserData Struct:
  + *Purpose*: The **UserData** struct serves as the container for storing all user-specific information within the budget tracker application. It contains the following crucial fields:
    - **username**: A string field that uniquely identifies each user. It is used for authentication and data isolation.
    - **password**: Another string field that securely stores the user's password for authentication.
    - **monthlyIncome**: A floating-point field representing the user's monthly earnings.
    - **rentAmount**, **taxesAmount**, **billsAmount**: Fields for monthly expenses specific to categories like rent, taxes, and bills.
    - **maxFoodBudget**, **maxEntertainmentBudget**, **maxClothesBudget**: These fields represent suggested maximum spending amounts for particular expense categories.
    - **incomeTransactions**: A vector of **Transaction** structs that records income entries.
    - **expenseTransactions**: Another vector of **Transaction** structs to record expense entries.
  + *Why This Choice*: The selection of the **UserData** struct is deliberate and offers several benefits:
    - **Data Encapsulation and Organization**: It encapsulates all user-specific information within a single structure, making it easier to manage, update, and ensure data isolation.
    - **Modularity and Scalability**: The struct allows for modular data handling and supports scalable user data management. Each user's data is stored as an instance of this struct, making it easy to add or retrieve information.
    - **Security**: By including the **password** field, the struct isolates sensitive user data, enhancing security. Sensitive information, such as passwords, should always be isolated and protected.

3. File Input/Output Functions:

* **loadUserData Function:**
  + *Purpose*: The **loadUserData** function is responsible for loading a user's data from a file, populating the **UserData** structure, and returning it. This function is crucial because it ensures that a user's financial data is retrieved when they log in.
  + *Key Features*:
    - The function takes the user's **username** as an input parameter, forming the filename (e.g., "username.txt") for data retrieval.
    - It opens and reads the user's data file, extracting information such as the username, password, monthly income, expense categories, and transaction history.
    - The function carefully parses the file's content to populate the relevant fields in the **UserData** structure.
    - If the user's file doesn't exist, it returns a status indicating that the file is not found.
  + *Importance*: The **loadUserData** function is essential for several reasons:
    - **Data Retrieval**: It ensures that a user's financial data is accessible when they log in. This enables the budget tracker to provide accurate and up-to-date financial information to the user.
    - **Data Isolation**: Each user's data is stored in their respective file, ensuring data isolation and privacy. This design choice enhances security and data management.
    - **User Experience**: It enhances the user experience by allowing users to pick up where they left off, viewing their transaction history and financial status.
* **saveUserData Function:**
  + *Purpose*: The **saveUserData** function is responsible for saving a user's data to a file, ensuring that their financial information is persisted for future use. This function is critical for preserving user data.
  + *Key Features*:
    - The function takes the user's **username** and their **UserData** structure as input parameters.
    - It opens and writes to the user's data file, storing information such as the username, password, monthly income, expense categories, and transaction history.
    - The function carefully formats and writes the data to the file, ensuring it can be loaded accurately during future sessions.
  + *Importance*: The **saveUserData** function is essential for several reasons:
    - **Data Persistence**: It ensures that a user's financial data is saved securely for future access. Users can continue to track their budget and transactions across different sessions.
    - **Data Isolation**: Just as with the **loadUserData** function, the **saveUserData** function maintains data isolation, preventing the mixing of user data.
    - **Security**: It contributes to data security by preserving the user's data on their local device. Sensitive information, such as passwords, should be stored securely and isolated from other users' data.

**Why These Choices**:

* **Data Isolation**: The choice to use individual files for each user's data ensures that each user's information is isolated, enhancing privacy and security. Users can only access their own data.
* **Data Persistence**: The decision to save and load data from files is practical for ensuring that user data is persisted across multiple sessions. This is particularly important for a budget tracker, where users expect continuity in managing their finances.
* **Security**: Storing the user's password in their data file ensures that sensitive data is stored securely. Passwords should never be stored in plain text or shared among users.

4. User Data Retrieval and Management:

* **findUserData Function:**
  + *Purpose*: The **findUserData** function plays a critical role in locating and retrieving a specific user's data based on their username. This function is essential for user authentication and data access.
  + *Key Features*:
    - The function takes the user's **username** as an input parameter.
    - It iterates through the **usersData** vector, which stores the **UserData** structures of all registered users, searching for a matching **username**.
    - If a matching user is found, a pointer to their **UserData** structure is returned. If no match is found, a **nullptr** is returned.
  + *Importance*: The **findUserData** function serves several important purposes:
    - **User Authentication**: It verifies the user's identity by checking if their username exists in the **usersData** vector. This is crucial for ensuring that only registered users can access their data.
    - **Data Retrieval**: When a user logs in, the function retrieves the user's specific data, enabling the budget tracker to provide personalized financial information and transaction history.
    - **Data Isolation**: It ensures data isolation by locating the user's data within the vector, preventing access to other users' information.

**Why This Choice**:

* **User Authentication**: User authentication is a fundamental security measure. The **findUserData** function verifies the user's identity, ensuring that only authorized users can access their data. This is vital for protecting user privacy and data.
* **Data Isolation**: By searching for and returning a specific user's data, the function maintains data isolation. It prevents unauthorized access to other users' information and ensures that each user's data is treated separately

5. Adding Income and Expenses:

* **addIncome Function:**
  + *Purpose*: The **addIncome** function allows users to input income transactions by specifying a description and the corresponding amount. This function is essential for tracking money earned.
  + *Key Features*:
    - The function prompts the user to enter a description for the income transaction (e.g., "Salary" or "Freelance Payment").
    - It then prompts the user to input the monetary amount for the income.
    - The entered income transaction is added to the user's **incomeTransactions** vector within the **UserData** structure.
  + *Importance*: The **addIncome** function serves several vital purposes:
    - **User Input**: It provides a user-friendly interface for inputting income transactions. This simplifies the process of recording money earned.
    - **Transaction Record**: It ensures that all income transactions are accurately recorded within the user's data, maintaining a comprehensive transaction history.
    - **Financial Insights**: By tracking income, the budget tracker can calculate and display the user's current balance and offer financial insights.
* **addExpense Function:**
  + *Purpose*: The **addExpense** function allows users to input expense transactions, including a description, amount, and category (e.g., "Food," "Rent," "Entertainment"). This is crucial for monitoring spending.
  + *Key Features*:
    - The function prompts the user to enter a description for the expense transaction (e.g., "Groceries" or "Movie Tickets").
    - Users are asked to input the monetary amount for the expense.
    - The function then guides users to select a category for the expense. Users can choose from predefined categories like "Food," "Rent," or "Entertainment," or create a custom category if necessary.
    - The entered expense transaction is added to the user's **expenseTransactions** vector within the **UserData** structure.
  + *Importance*: The **addExpense** function has several critical roles:
    - **Expense Tracking**: It allows users to record and categorize their expenses, which is vital for budget management and financial awareness.
    - **Category-Based Budgeting**: Categorizing expenses by category (e.g., "Food," "Rent") facilitates budgeting and spending analysis. It helps users understand where their money is going.
    - **Data Organization**: By adding expenses to the **expenseTransactions** vector, the function ensures that all expense transactions are structured and retrievable.

**Why These Choices**:

* **User-Friendly Input**: The choice to use these functions for income and expense input simplifies the user experience. Users can easily describe transactions and provide financial details.
* **Transaction Recording**: Recording income and expenses in structured vectors ensures that financial data is well-organized and can be used to calculate the user's financial balance and offer insights.
* **Category-Based Expense Tracking**: Categorizing expenses is an essential practice for effective budgeting and expense analysis. Users can understand where they spend the most and where potential savings can be made.
* **Data Isolation**: Each user's data is isolated within their **UserData** structure, ensuring that their income and expenses do not interfere with other users data.

6. Managing Additional Expenses and Budget Adjustment:

* **manageExpenses Function:**
  + *Purpose*: The **manageExpenses** function offers users the ability to manage additional expenses beyond their fixed monthly bills and initial expenses. This function is important for accommodating unexpected or variable costs and adjusting budget recommendations accordingly.
  + *Key Features*:
    - The function begins by providing users with suggested maximum monthly spending amounts for specific expense categories (e.g., "Food," "Entertainment," "Clothes"). These suggestions are based on the user's initial budget and remaining income after accounting for fixed expenses.
    - Users are given the choice to add additional expenses. If they choose to do so, the function guides them through the process of entering descriptions, amounts, and categorizing these expenses.
    - After adding additional expenses, the function recalculates and updates the suggested maximum spending limits for each category based on the remaining income.
    - Users are presented with the revised suggested maximum spending limits.
  + *Importance*: The **manageExpenses** function serves several important purposes:
    - **Adaptability**: It allows users to adapt their budget to account for unexpected or new expenses that may arise during the month. This is crucial for financial flexibility.
    - **Budget Adjustments**: The function recalculates suggested spending limits based on the updated expenses, ensuring that users receive relevant budget recommendations.
    - **User Engagement**: By providing insights into budget adjustments and remaining income, it encourages users to actively manage their finances and make informed spending decisions.

**Why This Choice**:

* **Adaptability and Flexibility**: This function recognizes the dynamic nature of personal finances. By allowing users to add and manage additional expenses, it accommodates real-world scenarios where budgets may need to be adjusted.
* **User-Centric**: The user's financial experience is prioritized by providing them with control over their budget. Users can make choices based on their unique circumstances.
* **Financial Insights**: The function enhances financial awareness by illustrating the impact of additional expenses on the remaining budget. It guides users in making informed spending decisions.
* **Data Isolation**: As with other functions, this function maintains data isolation, ensuring that each user's budget and additional expenses are treated separately.

7. User Registration (registerUser Function):

* **registerUser Function:**
  + *Purpose*: The **registerUser** function allows new users to create an account within the budget tracker application. This function is crucial for expanding the user base and establishing unique user profiles.
  + *Key Features*:
    - The function guides new users through the registration process, requesting them to input their desired username and password.
    - Users are prompted to provide details about their monthly income, monthly rent amount, monthly taxes amount, and monthly bills amount.
    - Based on these inputs, the function calculates and sets initial suggested maximum spending limits for expense categories (e.g., "Food," "Entertainment," "Clothes").
    - The user's data is saved to a file using their chosen username as the filename.
  + *Importance*: The **registerUser** function serves several key purposes:
    - **User Onboarding**: It facilitates the onboarding of new users, enabling them to create profiles and start using the budget tracker.
    - **Personalized Budgeting**: By collecting financial details, it allows the application to provide personalized budget recommendations and spending limits.
    - **Data Persistence**: The user's data is saved securely, ensuring that their information is available for future use.

8. User Login (login Function and Main Program Loop):

* **login Function:**
  + *Purpose*: The **login** function is used to authenticate existing users. It verifies user credentials by checking the provided username and password against the registered user data.
  + *Key Features*:
    - The function prompts users to enter their username and password.
    - It then searches for a matching user by iterating through the **usersData** vector.
    - If a match is found, a pointer to the user's **UserData** structure is returned, granting them access to their budget and transactions. If no match is found, a **nullptr** is returned, indicating a failed login attempt.
* **Main Program Loop**:
  + The main program loop is responsible for managing user interactions. It provides a menu of options to users, including registration, login, and various budget-related actions (e.g., adding income, adding expenses, calculating balance, managing expenses, and logging out).

**Importance and Significance of Choices**:

* **User Registration**: The choice to provide user registration is pivotal for expanding the application's user base. New users can create accounts and personalize their budget tracking experience.
* **User Authentication**: The login function ensures that only registered users can access their data. This is a fundamental security measure, safeguarding user privacy.
* **Personalization**: Collecting initial financial details during registration allows the application to personalize budget recommendations, making them more relevant to the user's financial situation.
* **Data Persistence**: Both registration and login choices contribute to data persistence. Registered user data is saved securely, ensuring it can be retrieved and used for future budget tracking sessions.
* **User Engagement**: The main program loop and menu-based interface engage users by providing them with options and actions related to budget management.
* **Data Isolation**: Each user's data is treated separately, maintaining privacy and security. User data is isolated in their respective files and structures.

9. Calculate and Display the Current Balance (Option 3):

* **Purpose**:
  + The "Calculate and Display Balance" option allows users to obtain an overview of their financial situation by calculating and displaying their current balance. This is one of the fundamental features of the budget tracker, providing users with insights into their financial health.
* **Key Features**:
  + When the user selects this option, the **calculateAndDisplayBalance** function is called.
  + This function calculates the total income and total expenses based on the user's recorded transactions.
  + It then computes the current balance by subtracting total expenses from total income.
  + The balance, along with total income and total expenses, is displayed to the user, allowing them to see the financial snapshot of their budget.
* **Importance**:
  + The "Calculate and Display Balance" feature is vital for several reasons:
    - **Financial Awareness**: It provides users with a clear understanding of their financial situation, helping them make informed decisions.
    - **Budget Evaluation**: Users can assess whether they are overspending or staying within their budget, allowing for necessary adjustments.
    - **User Engagement**: This feature encourages users to actively manage their finances and maintain a sense of control.

10. User Logout (Option 5):

* **Purpose**:
  + The "Logout" option allows users to safely end their session and protect their data. This is an important function to ensure user privacy and security.
* **Key Features**:
  + When the user selects this option, their current user data is saved to a file, securing their financial history.
  + A message is displayed to confirm that the user has been logged out successfully.
  + The user is then returned to the main program menu, where they can choose to log in again, register a new account, or exit the program.
* **Importance**:
  + User logout is crucial for the following reasons:
    - **Data Security**: It ensures that the user's financial data is saved before logging out, preventing data loss and ensuring privacy.
    - **User Control**: Users can safely end their sessions, preventing unauthorized access to their accounts.
    - **User Convenience**: This option offers a smooth transition between different users or sessions within the application.

**Why These Choices**:

* **Financial Awareness**: Calculating and displaying the balance is essential for users to remain aware of their financial situation and make informed financial decisions.
* **Data Security**: User logout ensures that user data is saved securely, preventing data loss and unauthorized access.
* **User Engagement**: These features encourage users to actively engage with the application, promoting financial responsibility.
* **Data Isolation**: Each user's data is isolated and saved independently, ensuring data privacy and separation.

11. Data Persistence (File I/O and saveUserData Function):

* **Purpose**:
  + Data persistence in the budget tracker application ensures that user data, including income, expenses, and budget settings, is saved to files, allowing users to resume their budget tracking from where they left off in the next session.
* **Key Features**:
  + The **saveUserData** function is responsible for saving user data to files. It takes the user's chosen username as a parameter and creates a file with that username as the filename.
  + User data, including username, password, income, expenses, and budget settings, is saved in the designated file.
  + This function is called in various scenarios, such as when a new user is registered, when a user logs out, or when additional expenses are added and the budget is updated.
* **Importance**:
  + Data persistence is crucial for the following reasons:
    - **User Convenience**: It allows users to continue budget tracking seamlessly in subsequent sessions.
    - **Data Security**: User data is preserved and secured in files, preventing data loss.
    - **User Engagement**: Data persistence encourages users to use the application consistently and achieve long-term financial goals.

12. User Logout and Main Program Loop:

* **Purpose**:
  + The combination of user logout and the main program loop allows users to efficiently manage their budget tracking sessions, switch between users, and exit the program when needed.
* **Key Features**:
  + When a user selects the "Logout" option, their data is saved using the **saveUserData** function, and they receive a confirmation message before returning to the main program menu.
  + In the main program loop, users can navigate between various options, including registration, login, budget actions, and logout.
  + The program loop ensures that user sessions are well-structured and user data remains isolated.
* **Importance**:
  + The combination of user logout and the main program loop is essential for the following reasons:
    - **Data Security**: User data is saved before logout, ensuring its safety and privacy.
    - **User Convenience**: The program loop offers a menu-based interface that allows users to perform various actions and seamlessly manage their budget tracking sessions.
    - **User Engagement**: Users can engage with the application comfortably, making the budget tracker a user-centric tool for financial management.

**Why These Choices**:

* **Data Security and Continuity**: Data persistence ensures that user data is secure and that users can pick up where they left off, enhancing user trust in the application.
* **User Engagement**: The combination of logout and the program loop provides a seamless and engaging user experience, encouraging users to actively manage their finances.
* **Data Isolation**: User data is stored independently, preserving privacy and preventing data overlap.

## <Code>

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include <iomanip>

using namespace std;

// Structure to represent income and expense transactions

struct Transaction {

    string description;

    double amount;

    string category; // Only for expenses

};

// Structure to store user data

struct UserData {

    string username;

    string password;

    double monthlyIncome;

    double rentAmount;

    double taxesAmount;

    double billsAmount;

    double maxFoodBudget;

    double maxEntertainmentBudget;

    double maxClothesBudget;

    vector<Transaction> incomeTransactions;

    vector<Transaction> expenseTransactions;

};

// Global vectors to store user data

vector<UserData> usersData;

// Function to load user data from a file and populate the UserData structure

bool loadUserData(const string& username, UserData& userData) {

    ifstream file(username + ".txt"); // Save in the same directory as the source code

    if (!file.is\_open()) {

        return false; // File doesn't exist

    }

    string line;

    while (getline(file, line)) {

        if (line.find("Username: ") == 0) {

            userData.username = line.substr(10);

        }

        else if (line.find("Password: ") == 0) {

            userData.password = line.substr(10);

        }

        else if (line.find("MonthlyIncome: ") == 0) {

            userData.monthlyIncome = stod(line.substr(15));

        }

        else if (line.find("RentAmount: ") == 0) {

            userData.rentAmount = stod(line.substr(12));

        }

        else if (line.find("TaxesAmount: ") == 0) {

            userData.taxesAmount = stod(line.substr(13));

        }

        else if (line.find("BillsAmount: ") == 0) {

            userData.billsAmount = stod(line.substr(12));

        }

        else if (line.find("MaxFoodBudget: ") == 0) {

            userData.maxFoodBudget = stod(line.substr(15));

        }

        else if (line.find("MaxEntertainmentBudget: ") == 0) {

            userData.maxEntertainmentBudget = stod(line.substr(23));

        }

        else if (line.find("MaxClothesBudget: ") == 0) {

            userData.maxClothesBudget = stod(line.substr(17));

        }

        else if (line == "IncomeTransactions:") {

            while (getline(file, line) && !line.empty()) {

                string description = line;

                getline(file, line);

                double amount = stod(line);

                userData.incomeTransactions.push\_back({ description, amount });

            }

        }

        else if (line == "ExpenseTransactions:") {

            while (getline(file, line) && !line.empty()) {

                string category = line;

                while (getline(file, line) && !line.empty()) {

                    string description = line;

                    getline(file, line);

                    double amount = stod(line);

                    userData.expenseTransactions.push\_back({ description, amount, category });

                }

            }

        }

    }

    file.close();

    return true;

}

// Function to save user data to a file

void saveUserData(const string& username, const UserData& userData) {

    ofstream file(username + ".txt"); // Save in the same directory as the source code

    if (!file.is\_open()) {

        cout << "Error: Cannot save user data.\n";

        return;

    }

    file << "Username: " << userData.username << endl;

    file << "Password: " << userData.password << endl;

    file << "MonthlyIncome: " << userData.monthlyIncome << endl;

    file << "RentAmount: " << userData.rentAmount << endl;

    file << "TaxesAmount: " << userData.taxesAmount << endl;

    file << "BillsAmount: " << userData.billsAmount << endl;

    file << "MaxFoodBudget: " << userData.maxFoodBudget << endl;

    file << "MaxEntertainmentBudget: " << userData.maxEntertainmentBudget << endl;

    file << "MaxClothesBudget: " << userData.maxClothesBudget << endl;

    file << "IncomeTransactions:" << endl;

    for (const auto& income : userData.incomeTransactions) {

        file << income.description << endl;

        file << income.amount << endl;

    }

    file << "ExpenseTransactions:" << endl;

    for (const auto& expense : userData.expenseTransactions) {

        file << expense.category << endl;

        file << expense.description << endl;

        file << expense.amount << endl;

    }

    file.close();

}

// Function to find a user by username

UserData\* findUserData(const string& username) {

    for (auto& user : usersData) {

        if (user.username == username) {

            return &user;

        }

    }

    return nullptr;

}

// Function to add income entries

void addIncome(UserData& userData) {

    Transaction income;

    cout << "Enter income description: ";

    cin >> income.description;

    cout << "Enter income amount: $";

    cin >> income.amount;

    userData.incomeTransactions.push\_back(income);

    cout << "Income added successfully!" << endl;

}

// Function to add expense entries

void addExpense(UserData& userData) {

    Transaction expense;

    cout << "Enter expense description: ";

    cin >> expense.description;

    cout << "Enter expense amount: $";

    cin >> expense.amount;

    cout << "Select expense category:\n";

    cout << "1. Healthcare\n2. Food\n3. Rent\n4. Taxes\n5. Bills\n6. Other\n";

    int categoryChoice;

    cin >> categoryChoice;

    switch (categoryChoice) {

    case 1:

        expense.category = "Healthcare";

        break;

    case 2:

        expense.category = "Food";

        break;

    case 3:

        expense.category = "Rent";

        break;

    case 4:

        expense.category = "Taxes";

        break;

    case 5:

        expense.category = "Bills";

        break;

    case 6:

        cout << "Enter custom category: ";

        cin.ignore(); // Clear the newline character from the previous input

        getline(cin, expense.category);

        break;

    default:

        cout << "Invalid category choice. Using 'Other'.\n";

        expense.category = "Other";

    }

    userData.expenseTransactions.push\_back(expense);

    cout << "Expense added successfully!" << endl;

}

// Function to calculate and display the current balance

void calculateAndDisplayBalance(const UserData& userData) {

    double totalIncome = 0.0;

    double totalExpenses = 0.0;

    for (const auto& income : userData.incomeTransactions) {

        totalIncome += income.amount;

    }

    for (const auto& expense : userData.expenseTransactions) {

        totalExpenses += expense.amount;

    }

    double balance = totalIncome - totalExpenses;

    cout << "Total Income: $" << fixed << setprecision(2) << totalIncome << endl;

    cout << "Total Expenses: $" << fixed << setprecision(2) << totalExpenses << endl;

    cout << "Current Balance: $" << fixed << setprecision(2) << balance << endl;

}

// Function to manage expenses

void manageExpenses(UserData& userData) {

    cout << "Suggested maximum monthly spending amounts based on your current budget:\n";

    cout << "Food: $" << fixed << setprecision(2) << userData.maxFoodBudget << endl;

    cout << "Entertainment: $" << fixed << setprecision(2) << userData.maxEntertainmentBudget << endl;

    cout << "Clothes: $" << fixed << setprecision(2) << userData.maxClothesBudget << endl;

    cout << "Do you want to add additional expenses? (yes/no): ";

    string addExpensesChoice;

    cin >> addExpensesChoice;

    if (addExpensesChoice == "yes") {

        double totalAdditionalExpenses = 0.0;

        while (true) {

            Transaction expense;

            cout << "Enter additional expense description (or 'done' to finish): ";

            cin >> expense.description;

            if (expense.description == "done") {

                break;

            }

            cout << "Enter additional expense amount: $";

            cin >> expense.amount;

            cout << "Select expense category:\n";

            cout << "1. Healthcare\n2. Food\n3. Rent\n4. Taxes\n5. Bills\n6. Other\n";

            int categoryChoice;

            cin >> categoryChoice;

            switch (categoryChoice) {

            case 1:

                expense.category = "Healthcare";

                break;

            case 2:

                expense.category = "Food";

                break;

            case 3:

                expense.category = "Rent";

                break;

            case 4:

                expense.category = "Taxes";

                break;

            case 5:

                expense.category = "Bills";

                break;

            case 6:

                cout << "Enter custom category: ";

                cin.ignore(); // Clear the newline character from the previous input

                getline(cin, expense.category);

                break;

            default:

                cout << "Invalid category choice. Using 'Other'.\n";

                expense.category = "Other";

            }

            userData.expenseTransactions.push\_back(expense);

            totalAdditionalExpenses += expense.amount;

        }

        // Update the budget based on additional expenses

        double remainingIncome = userData.monthlyIncome - userData.rentAmount - userData.taxesAmount - userData.billsAmount - totalAdditionalExpenses;

        userData.maxFoodBudget = remainingIncome \* 0.4; // 40% of remaining income

        userData.maxEntertainmentBudget = remainingIncome \* 0.3; // 30% of remaining income

        userData.maxClothesBudget = remainingIncome \* 0.3; // 30% of remaining income

        cout << "Updated suggested maximum monthly spending amounts:\n";

        cout << "Food: $" << fixed << setprecision(2) << userData.maxFoodBudget << endl;

        cout << "Entertainment: $" << fixed << setprecision(2) << userData.maxEntertainmentBudget << endl;

        cout << "Clothes: $" << fixed << setprecision(2) << userData.maxClothesBudget << endl;

    }

}

// Function to register a new user

void registerUser() {

    UserData newUser;

    cout << "Enter a username: ";

    cin >> newUser.username;

    cout << "Enter a password: ";

    cin >> newUser.password;

    cout << "Enter your monthly income: $";

    cin >> newUser.monthlyIncome;

    cout << "Enter your monthly rent amount: $";

    cin >> newUser.rentAmount;

    cout << "Enter your monthly taxes amount: $";

    cin >> newUser.taxesAmount;

    cout << "Enter your monthly bills amount: $";

    cin >> newUser.billsAmount;

    newUser.maxFoodBudget = newUser.monthlyIncome \* 0.4; // 40% of income

    newUser.maxEntertainmentBudget = newUser.monthlyIncome \* 0.3; // 30% of income

    newUser.maxClothesBudget = newUser.monthlyIncome \* 0.3; // 30% of income

    usersData.push\_back(newUser);

    saveUserData(newUser.username, newUser);

    cout << "User registered successfully!" << endl;

}

// Function to log in

UserData\* login() {

    string username, password;

    cout << "Enter your username: ";

    cin >> username;

    cout << "Enter your password: ";

    cin >> password;

    for (auto& user : usersData) {

        if (user.username == username && user.password == password) {

            return &user; // User found, return user data

        }

    }

    return nullptr; // User not found

}

int main() {

    while (true) {

        cout << "Options:\n";

        cout << "1. Register\n";

        cout << "2. Log In\n";

        cout << "3. Exit\n";

        cout << "Select an option: ";

        int choice;

        cin >> choice;

        switch (choice) {

        case 1:

            registerUser();

            break;

        case 2:

        {

            UserData\* currentUser = login();

            if (currentUser) {

                cout << "Logged in successfully!\n";

                while (true) {

                    cout << "\nUser Menu:\n";

                    cout << "1. Add Income\n";

                    cout << "2. Add Expense\n";

                    cout << "3. Calculate and Display Balance\n";

                    cout << "4. Manage Expenses\n";

                    cout << "5. Logout\n";

                    cout << "Select an option: ";

                    int userChoice;

                    cin >> userChoice;

                    switch (userChoice) {

                    case 1:

                        addIncome(\*currentUser);

                        break;

                    case 2:

                        addExpense(\*currentUser);

                        break;

                    case 3:

                        calculateAndDisplayBalance(\*currentUser);

                        break;

                    case 4:

                        manageExpenses(\*currentUser);

                        break;

                    case 5:

                        // Save user data before logging out

                        saveUserData(currentUser->username, \*currentUser);

                        cout << "Logged out successfully!\n";

                        break;

                    default:

                        cout << "Invalid option. Try again.\n";

                    }

                    if (userChoice == 5) {

                        break;

                    }

                }

            }

            else {

                cout << "Login failed. Username or password is incorrect.\n";

            }

        }

        break;

        case 3:

            return 0; // Exit the program

        default:

            cout << "Invalid option. Try again.\n";

        }

    }

}

## Results

