BUDGET TRACKER

Antonio, Christine Mae P.
Corro, Yaziniel
Doloso Jr. Andrew S.
Egar, Jarel Dave R.
Obias, Jave Joaquin C.

Technological Institute of the Philippines
Quezon City

November 2025

Table of Contents

Introduction

Many students today struggle with financial management due to a lack of clear goals, discipline, and motivation to save money. Poor budgeting and unmonitored spending often lead to unnecessary financial stress, which affects not only their academic performance but also their overall well-being. Without proper financial guidance or tools to help them manage their daily expenses, these individuals may experience repeated financial crises and find it difficult to recover from them. This problem highlights the growing need for innovative, technology-based solutions that can help promote accountability and financial awareness among young adults.

Several studies in recent years have examined the connection between financial literacy, self-control, and saving behaviour. Hani, Nasyalia, and Maya (2023) found that both financial literacy and financial inclusion have a significant positive impact on students' saving behaviour, suggesting that access to financial knowledge and services improves money-management habits. Similarly, Nesia and Sartika (2022) discovered that lifestyle and self-control play crucial roles in influencing saving behaviour among young workers, where greater discipline leads to more consistent saving patterns. These findings emphasize that developing financial awareness and self-control can reduce the likelihood of overspending and help individuals maintain stable finances.

However, many students continue to struggle with a lack of financial restrictions and uncontrolled spending habits, which worsen their financial challenges. Torres et al. (2024) found that college students often exhibit weak budgeting and mental accounting practices, making it difficult for them to adhere to spending limits or resist impulsive purchases. Likewise, Ablay (2023) reported that poor financial management skills and emotional spending habits contribute to students' inability to control their expenditures, leading to frequent budget shortfalls. These findings suggest that the absence of spending restrictions and self-discipline remains a major factor behind students' recurring financial instability.

To address these challenges, the programmers developed a program designed to help users track their daily expenses and establish better saving habits. The program allows users to monitor their spending and save money by setting a daily allowance for the week. If a user exceeds the limit for their allowance within the day, the excess amount will be deducted from the next day's allowance. Through this tool, students can gain greater control over their finances, reduce stress from overspending, and work toward long-term financial stability.

The Project

The Budget Tracker keeps track of the user's daily and weekly savings by calculating the amount left from the daily budget. Its secondary purpose is to encourage the user to not overspend through penalty. The penalty is the reduction of the next day's budget by the budget overrun on the current day. Additionally, the user can toggle the visibility of the recorded values.

To start, the user's credentials will be asked. Once the user has made an account or signed-up, they shall login. If the user is successfully logged in, the program will redirect to the main page. On the main page, there are choices: (1) Cash Out; (2) End Day; (3) Configure; (X) Exit. Initially, The program will track the budget based on two main inputs: the daily budget and the weekly goal savings amount. The daily budget is the amount that the user could use for the whole day. For option 1, If they used more than the budget, there will be a budget overrun and the next day's budget will be subtracted by the budget overrun; otherwise, the savings will simply be deducted by the amount used and will not affect the proceeding day's budget. The weekly goal savings amount will be determined whether it is reached or not by the end of 7 days, which is tracked by selecting option 2. Entering option 3 will redirect the program to the configure page, which has another set of options: (1) Change Budget; (2) Change Weekly Goal; (3) Savings Visibility; (4) Go Back; and (X) Exit. Option 1 and 2 will change the amount respectively. Option 3 determines whether to display the saved amount or keep it hidden. The 4th option will return the program to the main page, and selecting X will terminate the program. Similarly, the main page will terminate if the user input is X. If the weekly goal savings amount is reached, the program will display a congratulations message; otherwise, the program will inform that the goal was not reached.

Objectives

This project's main goal is to create a system that effectively tracks user savings, controls cash-out and ensures safe access via password protection in order to promote financial discipline.

- Admin system
- Penalty system if exceed amount of budget per day
- Goal tracking system that resets daily
- Show and hide savings

Flowchart of the System

<Flow Chart Subsection Introduction>

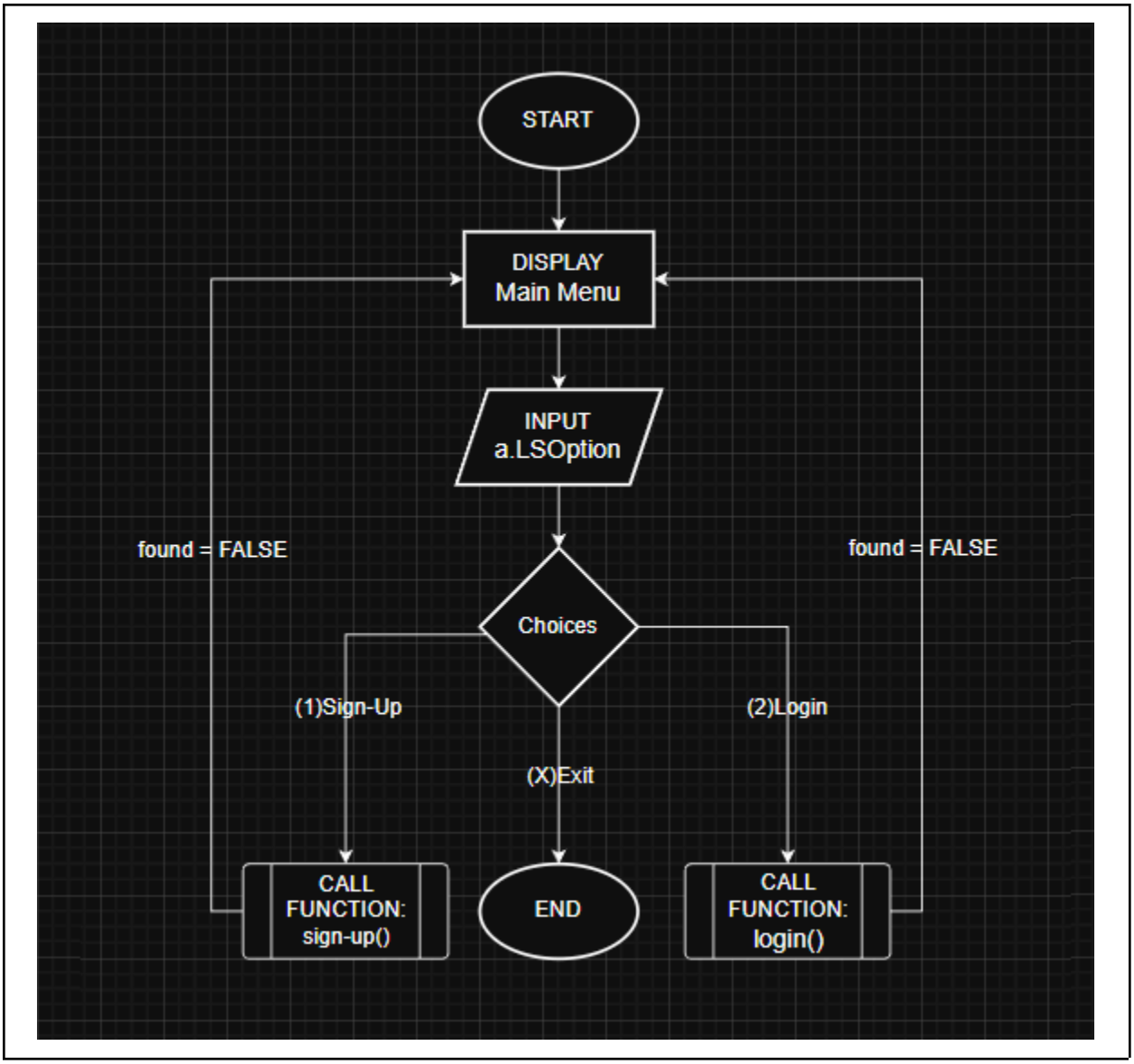


Figure 1: FUNCTION LSPage()

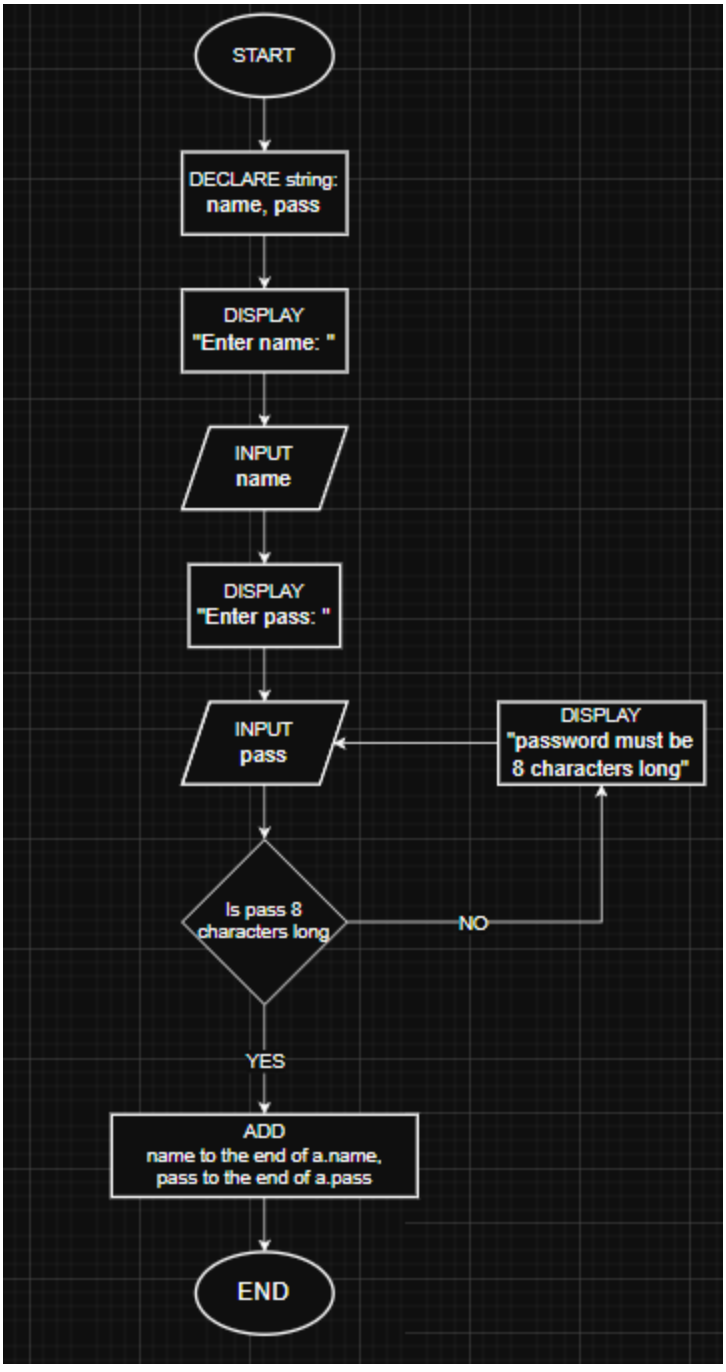


Figure 2: FUNCTION signup()

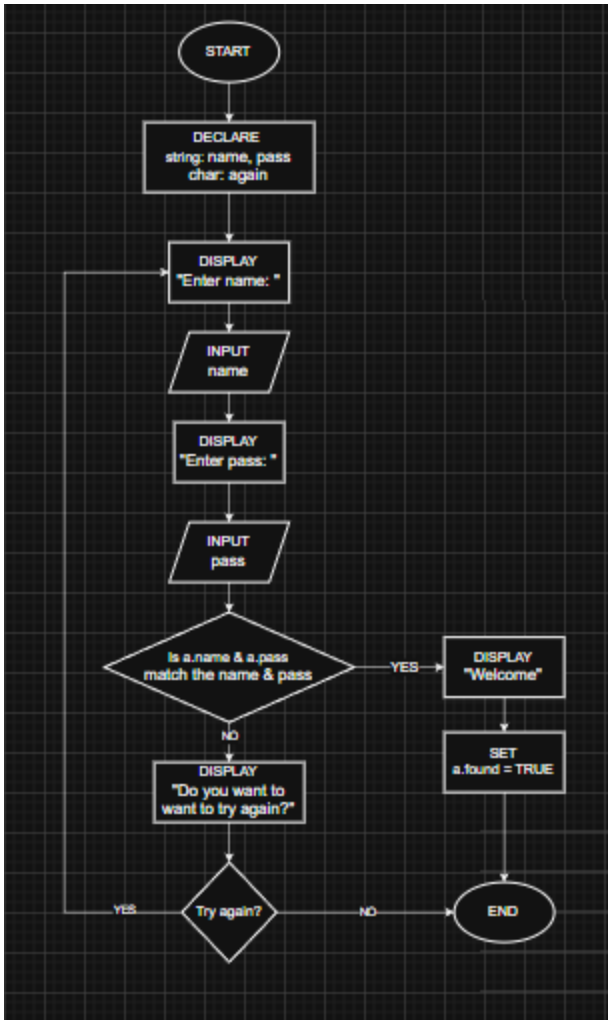


Figure 3: FUNCTION login()

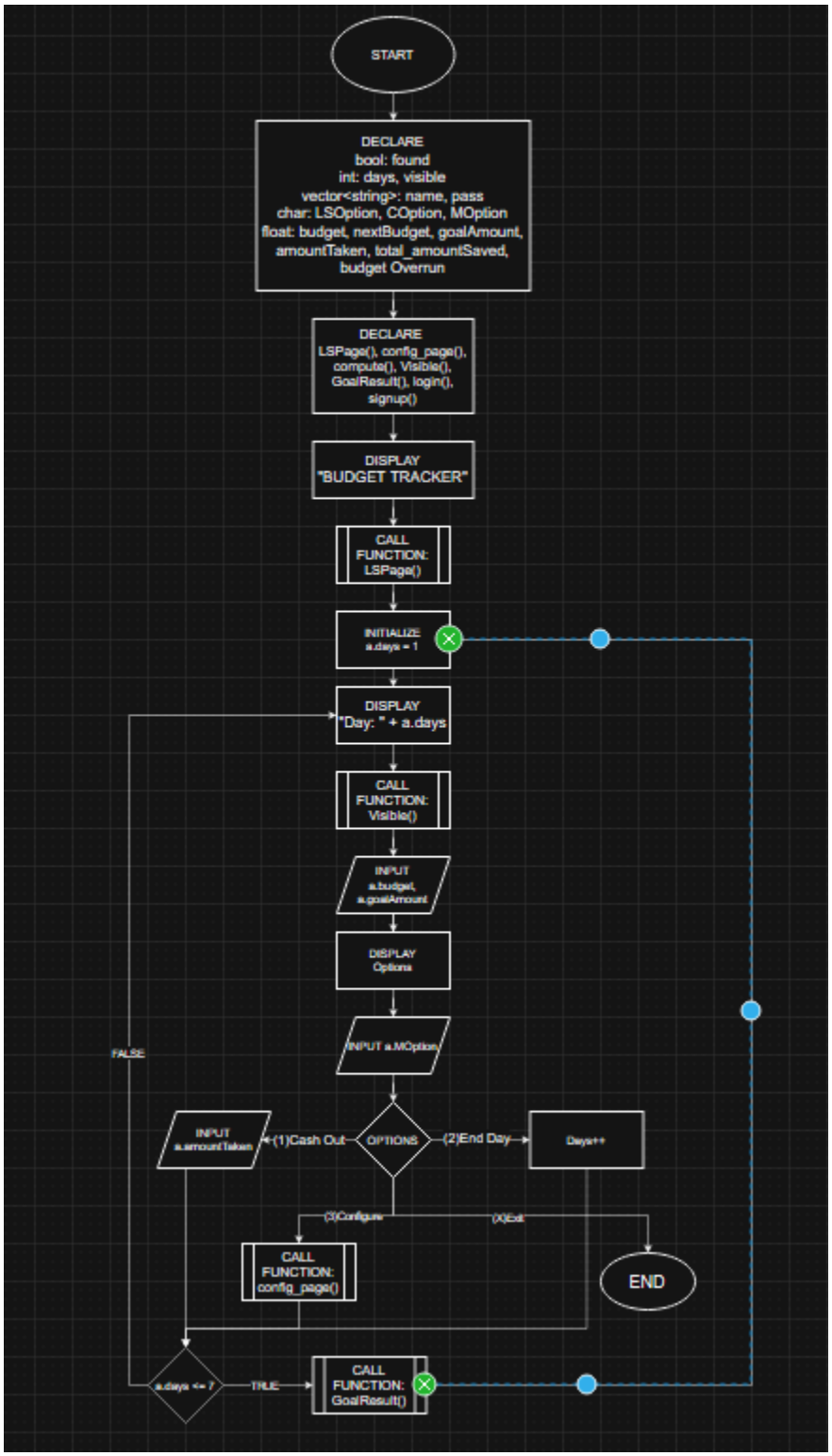
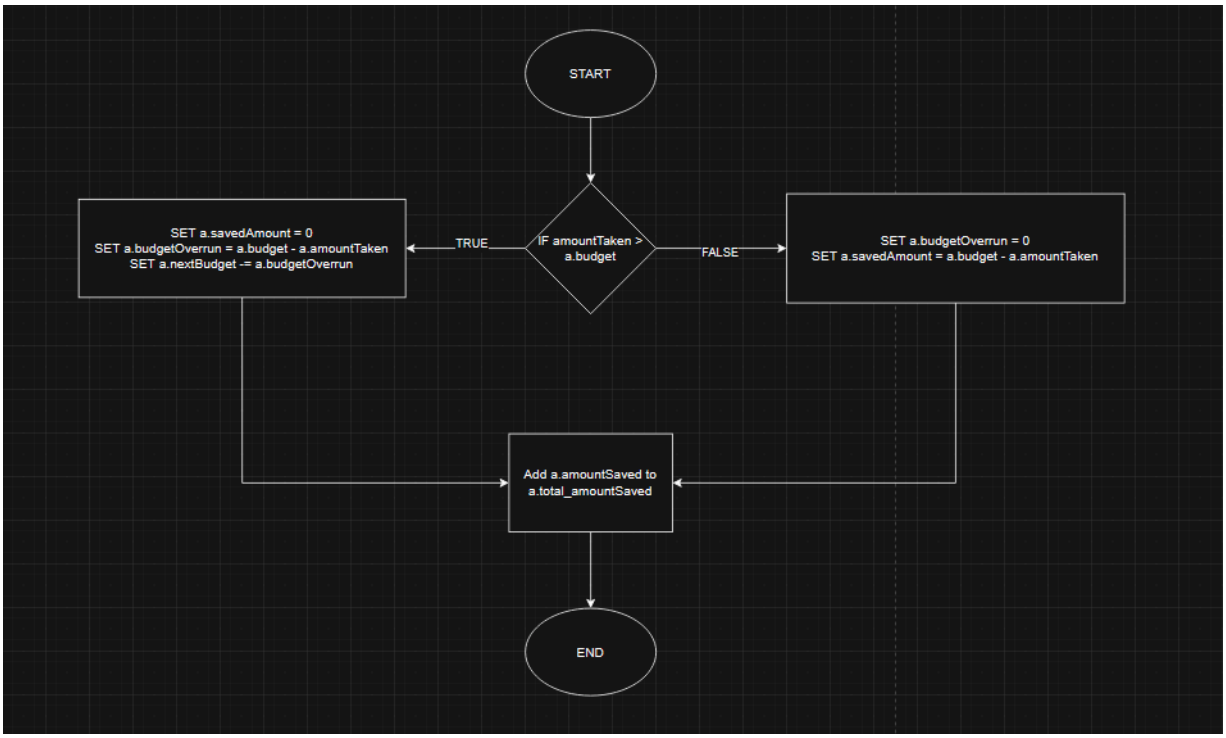
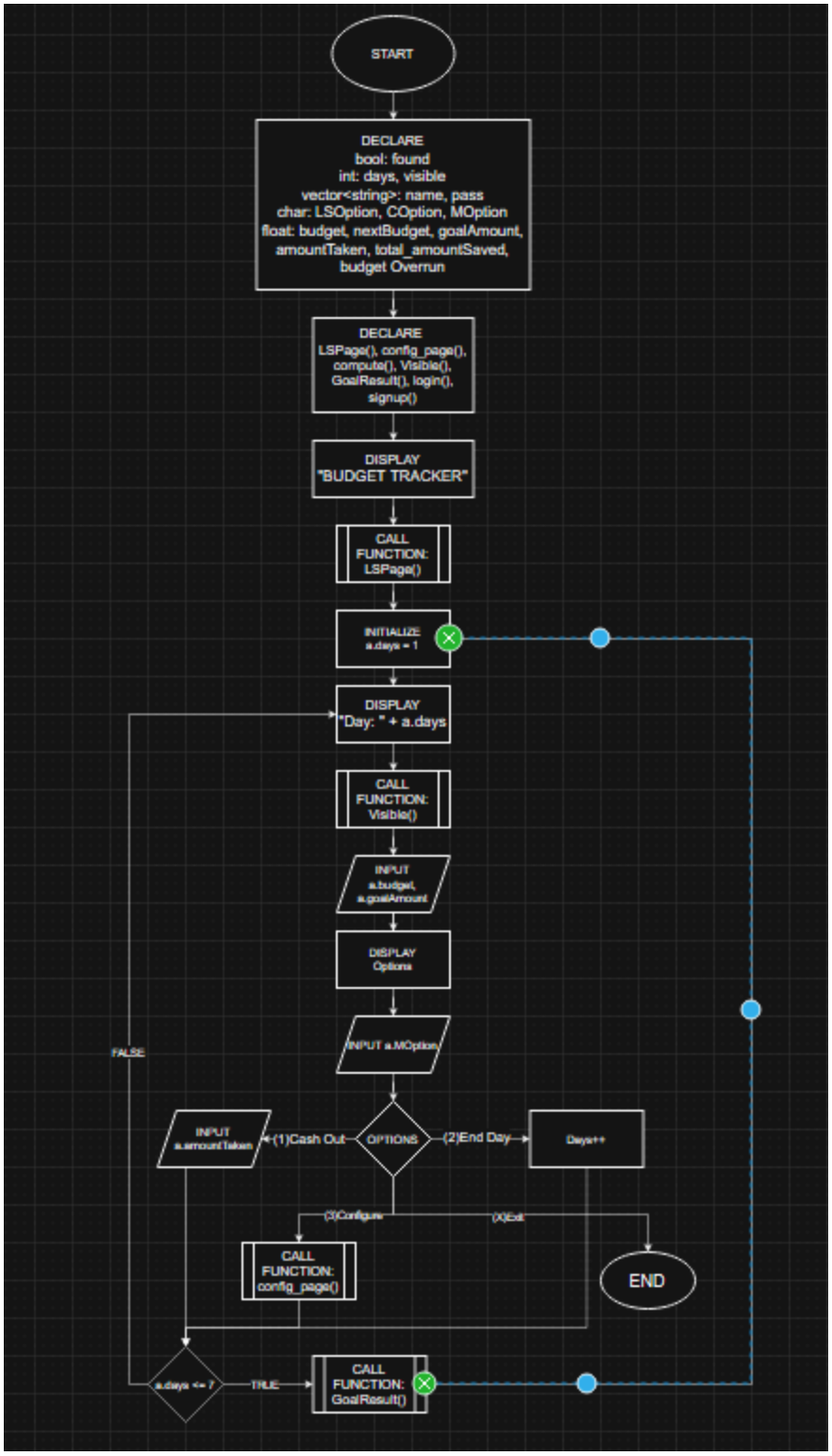


Figure 4: FUNCTION main()


START
IF amountTaken > a.budget
TRUE
SET a.savedAmount = 0
SET a.budgetOverrun = a.budget - a.amountTaken
SET a.nextBudget -= a.budgetOverrun
FALSE
SET a.budgetOverrun = 0
SET a.savedAmount = a.budget - a.amountTaken
Add a.amountSaved to a.total_amountSaved
END


START
DECLARE
bool: found
int: days, visible
vector<string>: name, pass
char: LSOption, COption, MOption
float: budget, nextBudget, goalAmount, amountTaken, total_amountSaved, budget Overrun
DECLARE
LSPage(), config_page(), compute(), Visible(), GoalResult(), login(), signup()
DISPLAY
"BUDGET TRACKER"
CALL FUNCTION: LSPage()
INITIALIZE
a.days = 1
DISPLAY
"Day: " + a.days
CALL FUNCTION: Visible()
INPUT
a.budget, a.goalAmount
DISPLAY
Options
INPUT a.MOption
FALSE
INPUT
a.amountTaken
(1)Cash Out
OPTIONS
(2)End Day
Days++
(3)Configure
(X)Exit
CALL FUNCTION: config_page()
END
a.days <= 7
TRUE
CALL FUNCTION: GoalResult()

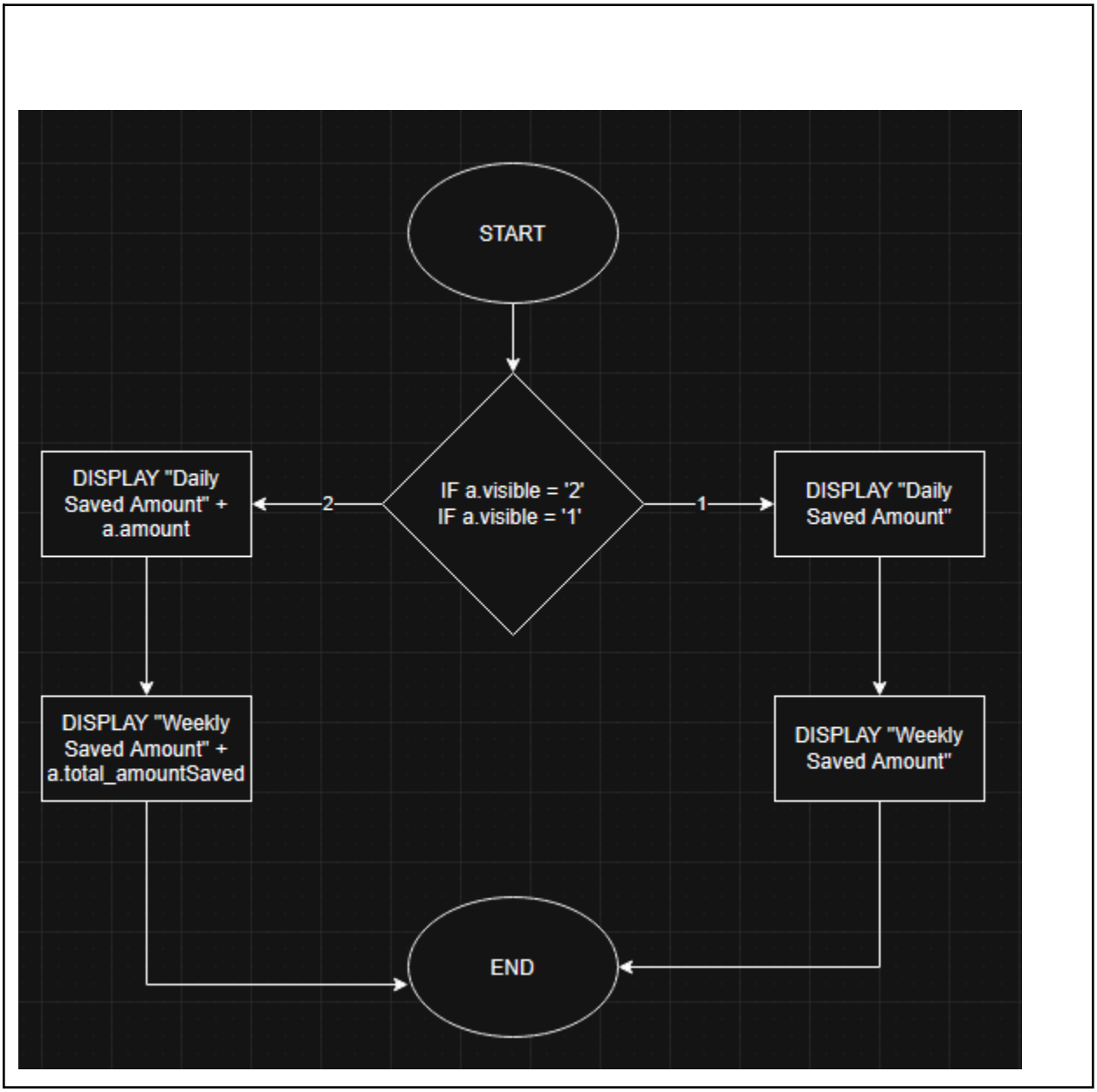


Figure 5: FUNCTION Visible()

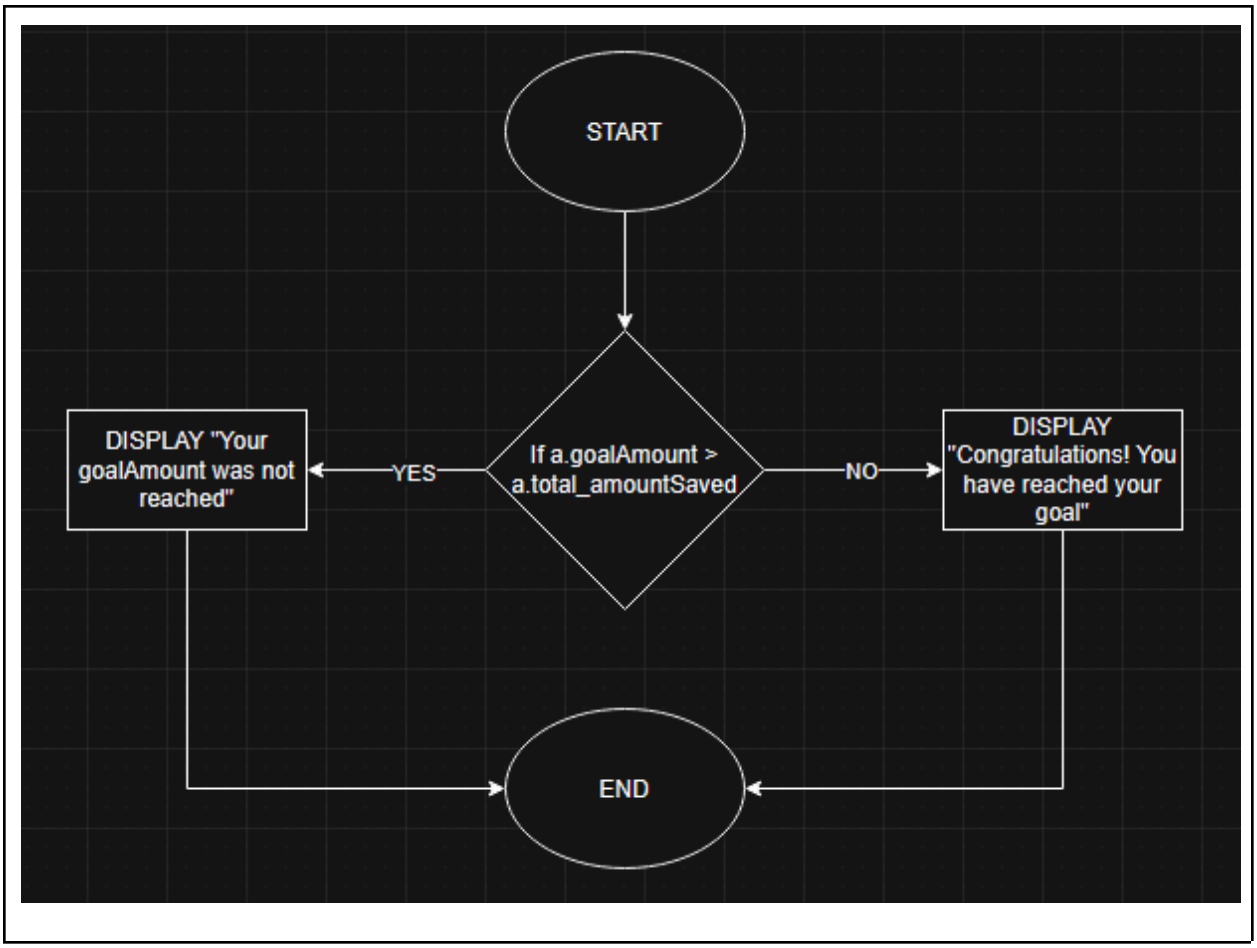


Figure 6: FUNCTION goalResult()

Pseudocode

This part is the overview of the program.

```
STRUCTURE tracker
        DECLARE char :  LSOption, COption, MOption
        DECLARE vector<string> : pass, name
        DECLARE float : budget, nextBudget, goalAmount, amountTaken,
        amountSaved,  total_amountTaken, total_amountSaved, budgetOverrun
        DECLARE int: days, visible
        DECLARE bool: found
END STRUCTURE
```

Figure 5: Structure tracker

This section defines a custom data structure, which organizes all the variables needed for the budget tracking system. It includes character options for navigation, vectors for storing user names and passwords, and multiple float variables to track budget metrics for savings, spending, and goals.

```
DECLARE:
 LSPage()
config_page()
compute()
Visible()
GoalResult()
login()
signup()

FUNCTION Title()
                DISPLAY  "Budget Tracker"
END FUNCTION
```

Figure 6: Declaration of functions and Title() Function

This part presents the core functions that drive the system's behavior, such as login, signup, configuration, and budget computations. Each function is declared for modularity and clarity, allowing the system to handle user input, display pages, and calculate financial outcomes.

```
FUNCTION main()
        CALL FUNCTION Title()
        CALL FUNCTION LSPage()
        INITIALIZE a.days = 1
        REPEAT
                REPEAT
                        DISPLAY "Day: " + a. days
                        CALL FUNCTION Visible()
                        GET a.budget, a.goalAmount
                        DISPLAY:
                        "(1) Cash Out
                         (2) End Day
                         (3) Configure
                         (X) Exit"
                        INPUT a.MOption
                        IF
                        a.MOption = 1 THEN
                        GET a.amountTaken
                        ELSE IF a.MOption = 2
                        Days++
                        ELSE IF a.MOption = 3
                        CALL FUNCTION config_page()
                        END IF
                UNTIL a.days <= 7
        CALL FUNCTION GoalResult()
        SET A.days = 1
        UNTIL LSOption != 'X'
END FUNCTION
```

Figure 7: main() function

This function is the main part of the program. It shows the title, login or signup page, and starts counting the days. Each day, it displays options like "Cash Out," "End Day," "Configure," or "Exit," and repeats this until all 7 days are done before showing if the savings goal was reached.

```
START FUNCTION LSPage()
REPEAT
                DISPLAY:
                (1) Sign-up
                (2) Login
                (X) Exit
                INPUT a.LSOption

                IF a.LSOption = 1 THEN
                                CALL FUNCTION signup()
                ELSE IF a.LSOption = 2 THEN
                                CALL FUNCTION login()
                ELSE IF a.LSOption = 'X' THEN
                                EXIT FUNCTION
                END IF
UNTIL found = FALSE
END FUNCTION
```

Figure 8: LSPage() Function

This is the login/signup page. It repeatedly shows a menu that lets the user sign up, log in, or exit. Depending on what the user chooses, it calls the correct function (signup or login).

```
START FUNCTION signup()
DECLARE string : name, pass
DISPLAY “Enter name: ”
INPUT name
REPEAT
        DISPLAY “Enter password: ”
        INPUT pass
        IF pass is not 8 characters long
                DISPLAY “Password must be 8 characters long”
        END IF
UNTIL pass is not 8 characters long
Add name to the end of a.name
Add pass to the end of a.pass
END FUNCTION
```

Figure 9 signup() function

This function is used when a new user wants to register. It asks for a name and password, checks that the password is exactly 8 characters long, and then stores the name and password once they are valid.

```
START FUNCTION login()
DECLARE string : name, pass
DECLARE char : again
IF a.name is empty THEN
        DISPLAY "No users yet."
        SET a.found = FALSE
        EXIT FUNCTION
END IF

REPEAT
        DISPLAY "Enter name: "
        INPUT name
        DISPLAY "Enter password: "
        INPUT pass

        IF a.name & a.pass match name and pass THEN
                DISPLAY "Welcome"
                SET a.found = TRUE
                EXIT FUNCTION
                ELSE IF a.name or a.pass has no match THEN
                        Ask User to try again (Y / N)
                        GET again
                        EXIT FUNCTION
                END IF
        END IF
UNTIL found = TRUE OR again = 'N'
END FUNCTION
```

Figure 10: login()

This part of function describes a log in info about the users and promp to enter user authentication. If the credentials match, it displays a welcome message. otherwise, it asks if the user wants to try again. The process repeats until a successful login or the user chooses not to continue.

```
START FUNCTION config_page()
                REPEAT
                DISPLAY:
                "(1) Change Budget
                 (2) Change Weekly Goal
                 (3) Savings Visibility
                 (4) Go Back
                 (X) Exit "
                INPUT a.COption

                IF a.COption = '1' THEN
                                GET a.budget
                ELSE IF a.COption = '2' THEN
                                GET a.goalAmount
                ELSE IF a.COption = '3' THEN
                                GET a.visible
                ELSE IF a.COption = 'X'
                                END PROGRAM
                END IF
                UNTIL a.COption != '4'
END FUNCTION
```

Figure 11: config_page() function

This is the settings or configuration page. The user can change the budget, weekly goal, or savings visibility here. It keeps running until the user chooses to go back or exit the program.

```
START FUNCTION compute()
        IF a.amountTaken > a.budget THEN
                SET a.savedAmount = 0
                SET a.budgetOverrun = a.budget - a.amountTaken
                SET a.nextBudget -= a.budgetOverrun
        ELSE
                SET a.budgetOverrun = 0
                SET a.savedAmount = a.budget - a.amountTaken
        END IF

        Add a.amountSaved to a.total_amountSaved

END FUNCTION
```

Figure 12: compute() function

This is the settings or configuration page. The user can change the budget, weekly goal, or savings visibility here. It keeps running until the user chooses to go back or exit the program.

```
START FUNCTION Visible()
        IF a.visible = '2' THEN
        DISPLAY "Daily Saved Amount: " + a.amountSaved
        DISPLAY "Weekly Saved Amount: " + a.total_amountSaved
        ELSE IF a.visible = '1' THEN
        DISPLAY "Daily Saved Amount: *"
        DISPLAY "Weekly Saved Amount: *"
END FUNCTION
```

Figure 13 Visible() function

This function controls how savings are shown. If visibility is set to "2," it displays the real savings amount; if set to "1," it hides the numbers using asterisks

```
START FUNCTION goalResult()
        IF a.goalAmount > a.total_amountSaved THEN
                DISPLAY "Your goalAmount was not reached"
        ELSE IF a.goalAmount < a.total_amountSaved THEN
                DISPLAY "Congratulations! You reached your goal savings
amount!"
        END IF
END FUNCTION
```

Figure 14: goalResult()

This function checks if the user has reached their savings goal. If not, it shows "Your goal was not reached," but if the goal is achieved, it displays "Congratulations! You reached your goal!"

Data Dictionary

The data dictionary provides an overview of the variables used in creating this project, organizing all the data used in developing the system. The data dictionary serves as a reference to gain a better understanding of how data is saved, organized, and changed during program execution. It also ensures consistency and clarity in data management, making the program easier to maintain and improve in the future.

The table below presents a detailed breakdown of all variables used in the program, especially in the tracker structure. It identifies the data name, size, data type, and the description of what each variable represents. This provides an organized overview of the framework of the system.

Table 1: Data Dictionary

Data Name	Size	Data Type	Description
1. LSOption	1 byte	char	Stores the user's choice in the login or signup menu
2. COption	1 byte	char	Stores the user's choice in the configuration menu
3. MOption	1 byte	char	Stores users' options in the main menu
4. Again	1 byte	char	if the user input for login.pass is incorrect, the program asks whether the user wants to try again or not. (Y) Yes or (N) No
5. Signup.pass	24 bytes	string	Stores passwords in the signup() function

6. Signup.name	24 bytes	string	Store username in the signup() function
7. Login.name	32 bytes	string	Store username in the login() function
8. Login.pass	32 bytes	string	Stores passwords in the login() function
9. name	32	string	Stores usernames that are entered and verified during the login() function.
10. pass	32	string	Stores password that are entered and verified during the login() function.
10. Budget	4 bytes	float	The “Current day” entered the budget
11. NextBudget	4	float	Stores the adjusted budget for the next day
12. Goalamount	4	float	Stores the weekly savings goal of the users
13. amountTaken	4	float	Stores the amount spent or “cashed out” for the day
14. amountSaved	4	float	Stores the amount saved for the current day
15. total_amountTaken	4	float	The total amount of all taken over the week
16. total_amountSaved	4	float	The total amount of all savings over the week
17. budgetOverrun	4	float	The overspent amount beyond the daily budget

18. Days	1	int	Tracks the current day in number (1-7)
19. Visible	1	int	Determines whether savings visibility is ON (2) or OFF (1)
20. found	1	bool	It indicates if the user was successfully logged in.

Code

This section provides an overview of the code of the system.

```
#include <iostream>
#include <string>
#include <vector>
#include<sstream>
using namespace std;

//VARIABLES
struct tracker {
    char LSOption, COption, MOption;
    vector<string> pass;
    vector<string> name;
    float budget, nextBudget, goalAmount, amountTaken, amountSaved,
    total_amountTaken, total_amountSaved, budgetOverrun;
    int days, visible;

    bool found;
};

//INITIALIZING FUNCTIONS
string LSPage(tracker &a); //shows the option to login or to signup
void config_page(tracker &a); //allows configuration of values
void compute(tracker &a); //calculation
void Visible(tracker &a); //outputs results based on visibility (found in config_page())
void GoalResult(tracker a); //outputs a message based on whether goal is achieved or not
void login(tracker &a);
void signup(tracker &a);


//TITLE
void Title() {
    cout << "Budget Tracker\n";
}
```

Figure 3. Structures and Variables

Code discussion

This section defines the tracker structure, which holds all user and budget-related information such as names, passwords, budget values, goals, and visibility settings. It also declares the prototypes of several functions used throughout the program for configuration, computation, and user interaction.

```
35  int main() {
36
37      tracker a;
38      Title();
39
40      cout << LSPage(a);
41
42      a.days = 1;
43
44      a.amountSaved = 0;
45      a.total_amountSaved = 0;
46      a.amountTaken = 0;
47      a.total_amountTaken = 0;
48      a.budget = 0;
49      a.nextBudget = 0;
50      a.budgetOverrun = 0;
51      a.visible = 2;
52
```

Figure 3. Main Page

The function initializes a main page and sets all its values to zero or default states. It also displays the program title and prepares the main loop for diaily and weekly budget

```
53      do {
54          do {
55          cout << endl << "=========================================\n";
56          cout << endl << "DAY: " << a.days  << endl;
57
58          Visible(a);
59          cout << endl;
60
61          if (a.budget <= 0) {
62
63                  cout << "Enter today's budget: ";
64                  cin >> a.budget ;
65                  a.nextBudget = a.budget;
66
67                  a.nextBudget += a.budgetOverrun;
68
69
70              if (a.days == 1) {
71              cout << "Enter weekly goal amount: ";
72              cin >> a.goalAmount;
73              }
74
75          }
```

Figure 3. Budget and Weekly Goal

This section manages the daily budget input and weekly goal setup. It prompts the user to enter a daily budget and goal amount for the week while updating the tracker's budget values accordingly.

Code discussion

```
77          stringstream s, s1;
78          s << a.amountSaved;
79          s1 << a.nextBudget;
80          string result = s.str();
81          string result1 = s1.str();
82
83
84          if (a.visible == 2) {
85              cout << "budget overrun: " << a.budgetOverrun << endl;
86              cout << "Today's adjusted budget: ";
87              if (a.budgetOverrun > 0) {
88                  cout << a.amountSaved << endl;
89              } else {
90                  cout << a.nextBudget << endl;
91              }
92          } else if (a.visible == 1) {
93              cout << "budget overrun: " << a.budgetOverrun << endl;
94
95              cout << "Today's adjusted budget: ";
96              if (a.budgetOverrun > 0) {
97              for (int i = 0; i < result.length(); i++) {
98                  cout << "*";
99              }
100             } else {
101                 for (int i = 0; i < result1.length(); i++) {
102                 cout << "*";
103                 }
104             }
105         }
```

Figure 3. Visible Function

Code discussion

The Visible() function displays the user's budget information depending on the visibility mode. It shows the adjusted budget and overrun data, using symbols or numbers based on the user's visibility settings.

```
108         cout << "\n(0) Cash Out\n";
109         cout << "(1) End Day\n";
110         cout << "(2) Configure\n";
111         cout << "(3) Sign-out\n";
112         cout << "(X) Exit\n";
113         cout << "Enter: ";
114         cin >> a.MOption;
115
116         switch (a.MOption) {
117             case '0':
118
119             cout << "Enter cash out amount: ";
120             cin >> a.amountTaken;
121
122             compute(a);
123
124             break;
125             case '1':
126
127             a.total_amountSaved += a.amountSaved;
128             compute(a);
129
130             a.days++;
131             a.budget = 0;
132             a.amountSaved = 0;
133
134             break;
135
136             case '2':
137                 config_page(a);
138             break;
139
140             case '3':
141             LSPage(a);
142             break;
143
144             case 'X':
145             exit(1);
146             break;
147         }
```

Figure 3. Main

Code discussion

This part provides the main menu options like cashing out, ending the day, configuring settings, or signing out. It uses a switch-case structure to handle each user command, updating or displaying data based on their choice.

```
149     } while (a.days <= 7);
150
151     GoalResult(a);
152
153     a.days = 1;
154
155     } while(a.MOption != 'X');
156
157     return 0;
158
159 }
```

Figure 3. Main

Code discussion

Figure 3. Main (Loop End):

This loop runs until the week (7 days) ends or until the user exits the program. After each cycle, it evaluates the user's goal progress and resets for the next week.

```
161  //LOGIN & SIGN-UP PAGE
162  string LSPage(tracker &a) {
163
164      do {
165          cout << endl;
166          cout << "(1) Sign-up\n";
167          cout << "(2) Login\n";
168          cout << "(X) Exit\n";
169          cout << "Enter: ";
170          cin >> a.LSOption;
171
172          if (a.LSOption != 'X') {
173
174              switch (a.LSOption) {
175
176              //SIGN UP
177              case '1':
178                  cout << endl;
179                  signup(a);
180
181              break;
182
183              //LOGIN
184              case '2':
185                  cout << endl;
186                  login(a);
187              break;
188
189              }
190
191          }
192
193      } while (a.found == false);
194
195      return "\nGoing to Main Page......\n";
196
197  }
```

Figure 3. LS Page

Code discussion

This function handles the login and signup interface, letting users choose whether to sign up, log in, or exit. It loops until a valid user is found or created, then transitions to the main page.

```
199 void signup (tracker &a) {
200
201     string name, pass;
202
203     do {
204         cout << "Enter name: ";
205         cin >> name;
206
207         cout << "Enter password: ";
208         cin >> pass;
209
210         if (pass.length() < 8) {
211             cout << endl << "Password must be at least 8 characters long." << endl;
212         }
213     } while (pass.length() < 8);
214
215     a.name.push_back(name);
216     a.pass.push_back(pass);
217
218     a.found = false;
219 }
```

Figure 3. Log in Info

Code discussion

The signup() function lets a new user register by entering a name and a password of at least 8 characters. The credentials are stored in the tracker structure, but login verification hasn't yet occurred.

```
221 void login (tracker &a) {
222
223     string name, pass;
224     char again;
225
226     //no existing account yet; haven't signed in
227     if (a.name.empty()) {
228         cout << "No Users yet.\n\n";
229         a.found = false;
230         return;
231     }
232
233     do {
234         cout << "Enter name: ";
235         cin >> name;
236
237         cout << "Enter password: ";
238         cin >> pass;
239
240         //if the name and password are existing
241         for (int i = 0; i < a.name.size(); i++) {
242             if (a.name[i] == name && a.pass[i] == pass) {
243                 cout << "Welcome\n";
244                 a.found = true;
245                 return;
246             }
247         }
248         cout << "The user either does not exist or password is incorrect. Try Again? (Y/N)\n";
249         cin >> again;
250
251         if (again == 'N') {
252             return;
253         }
254     } while (again == 'Y');
255 }
```

Figure 3. Log in Info

Code discussion

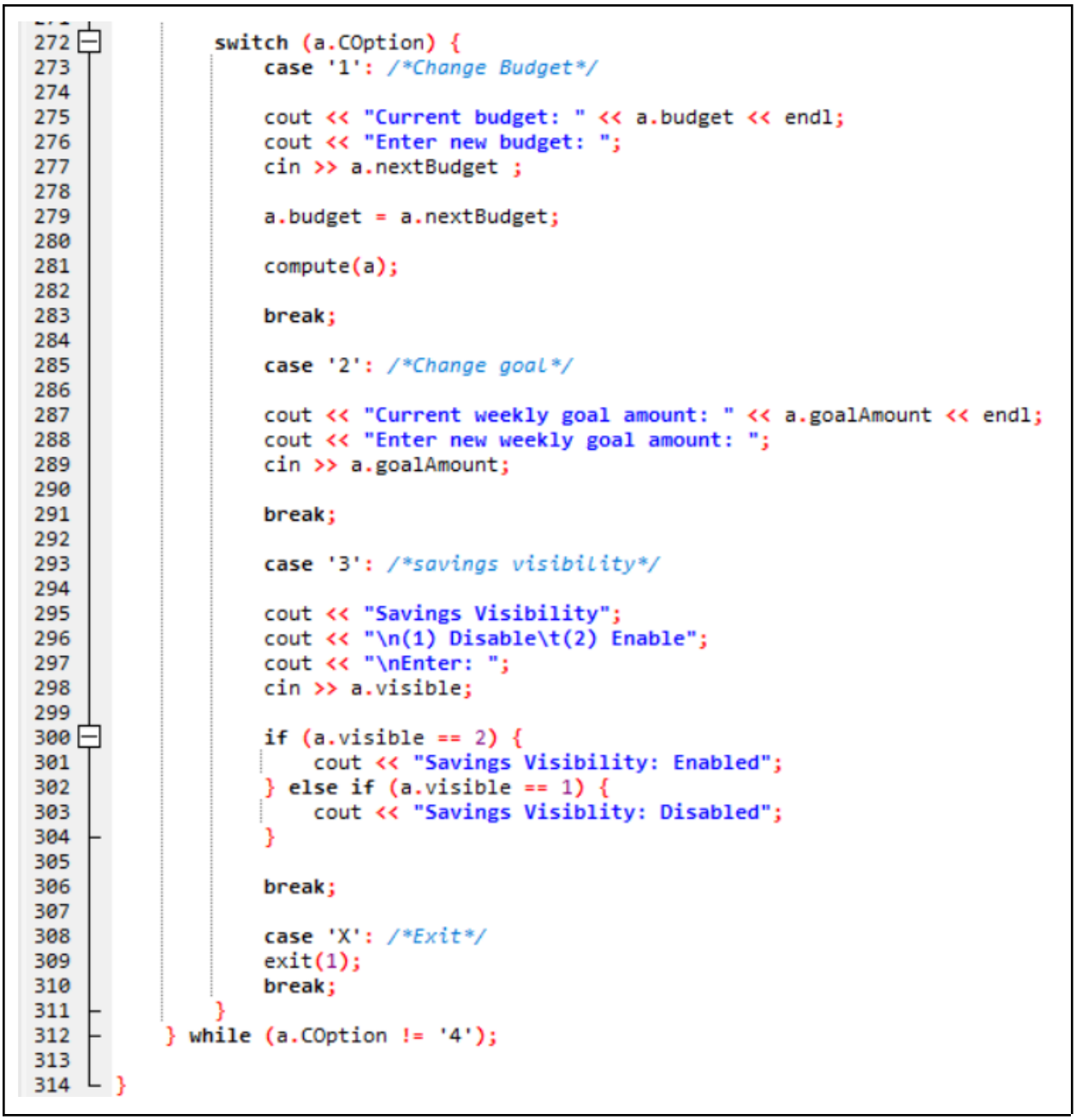
The login() function allows existing users to sign in by checking entered credentials against stored names and passwords. It loops until valid credentials are entered or the user exits.

```
257  //CONFIGURE PAGE
258  void config_page(tracker &a) {
259
260      do {
261
262          cout << endl << "=========================================\n";
263
264          cout << "(1) Change Budget\n";
265          cout << "(2) Change Weekly Goal\n";
266          cout << "(3) Savings Visibility: " << a.visible << endl;
267          cout << "(4) Go Back\n";
268          cout << "(X) Exit\n";
269          cout << "Enter: ";
270          cin >> a.COption;
271
```

Figure 3. Configure Page

Code discussion

The config_page() function provides a menu for adjusting budget settings, weekly goals, and visibility preferences. It allows users to update current values or exit back to the main menu.



```
        switch (a.COption) {
            case '1': /*Change Budget*/

            cout << "Current budget: " << a.budget << endl;
            cout << "Enter new budget: ";
            cin >> a.nextBudget ;

            a.budget = a.nextBudget;

            compute(a);

            break;

            case '2': /*Change goal*/

            cout << "Current weekly goal amount: " << a.goalAmount << endl;
            cout << "Enter new weekly goal amount: ";
            cin >> a.goalAmount;

            break;

            case '3': /*savings visibility*/

            cout << "Savings Visibility";
            cout << "\n(1) Disable\t(2) Enable";
            cout << "\nEnter: ";
            cin >> a.visible;

            if (a.visible == 2) {
                cout << "Savings Visibility: Enabled";
            } else if (a.visible == 1) {
                cout << "Savings Visiblity: Disabled";
            }

            break;

            case 'X': /*Exit*/
            exit(1);
            break;
        }
    } while (a.COption != '4');

}
```

Figure 3. Change Budget / Goal

Code discussion

This portion updates budget and goal settings and allows toggling visibility mode. Users can change daily and weekly financial parameters or enable/disable visibility to control how data is displayed.

```
326  //COMPUTE FUNCTION
327  void compute(tracker &a) {
328
329      // budget overrun
330      if (a.amountTaken >= a.nextBudget) {
331          a.budgetOverrun = a.budget - a.amountTaken;
332          a.nextBudget += a.budgetOverrun;
333          a.amountSaved = 0;
334
335      // amount cash out is within the budget
336      } else if (a.amountTaken < a.nextBudget) {
337
338          a.budgetOverrun = 0;
339
340          if (a.amountSaved <= 0) {
341              a.amountSaved = a.nextBudget - a.amountTaken;
342          } else if (a.amountSaved > 0) {
343              a.amountSaved -= a.amountTaken;
344          }
345      }
346  }
```

Figure 3. Computation

Code discussion

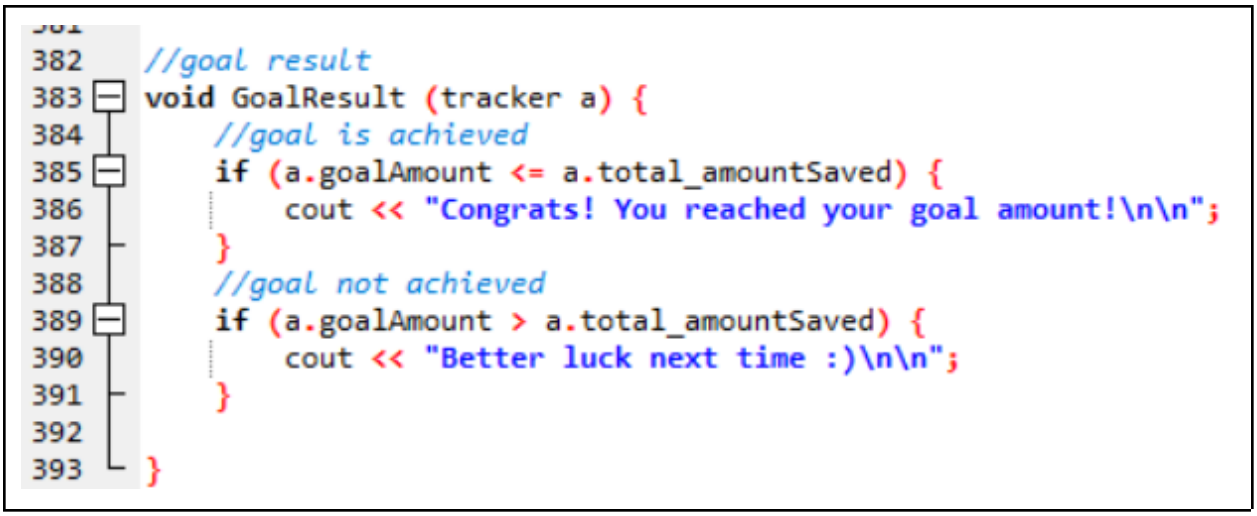
The compute() function manages budget calculations, detects overspending, and updates savings accordingly.

```
348  //show and hide savings
349  void Visible(tracker &a) {
350
351      //convert a.amountSaved from float to string
352      stringstream s;
353      s << a.amountSaved;
354      string result = s.str();
355
356
357      switch (a.visible) {
358          case 1: //hide values
359
360              cout << "Daily saved amount: ";
361
362              for (int i = 0; i < result.length(); i++) {
363                  cout << "*";
364              }
365
366              cout << "\nWeekly saved amount: ";
367
368              for (int i = 0; i <= a.total_amountSaved; i++) {
369              cout << "*";
370
371          }
372          break;
373          case 2: //show values
374
375              cout << "Daily saved amount: " << a.amountSaved << endl;
376              cout << "Weekly saved amount: " << a.total_amountSaved << endl;
377          break;
378      }
379
380  }
```

Figure 3. Show and Hide Savings

Code discussion

The Visible function manages the display of the user's savings by either showing the actual values or masking them with asterisks for privacy. It allows users to easily toggle the visibility of their financial information on screen.



```
382  //goal result
383  void GoalResult (tracker a) {
384      //goal is achieved
385      if (a.goalAmount <= a.total_amountSaved) {
386          cout << "Congrats! You reached your goal amount!\n\n";
387      }
388      //goal not achieved
389      if (a.goalAmount > a.total_amountSaved) {
390          cout << "Better luck next time :)\n\n";
391      }
392
393  }
```

Figure 3. Goal Result

Code discussion

The GoalResult() function checks if the user's savings goal is reached and provides appropriate feedback.

Results and Discussion

The Budget Tracker Program was designed to help users manage their daily budgets, monitor expenses (cash out), and accomplish their weekly savings or goals. When executed, the program allows users to sign up or log in, set daily budgets, record cash outs, and view the progress regarding their savings. It provides clear feedback to the user about his financial status each day including the daily and weekly amount saved and can be shown or hidden according to the visibility setting. At the end of the week, the program compares the total savings with the user's goal and displays an encouraging message, congratulating the user for achieving their weekly goal.

The program effectively performs its main functions, such as calculating savings, handling daily budgets, and displaying results. Its use of a structured data type (tracker) to store key information like budgets, expenses, and savings makes it easier to manage and debug. The modular design, with functions like compute(), Visible(), and GoalResult(), helps maintain a clear and organized code structure, supporting readability and future improvements.

However, testing has brought up two major issues. First, the overall total of savings for the week will only update if the user has done a “cash out” before closing the day; otherwise, the saved value is not added to the weekly total. Second, both the login and sign-up systems do not save information permanently. Everything goes into temporary memory and is then discarded upon termination of the program. The addition of file handling or database support can allow user data to persist between sessions.

In short, this Budget Tracker Program depicts a functional and well structured design that can handle basic financial tracking. While the core features work correctly, addressing these identified gaps in automatic savings updates and data persistence would make the system more reliable and realistic for long term personal budgeting applications.

```
Budget Tracker

(1) Sign-up
(2) Login
(X) Exit
Enter: 2

No Users yet.


(1) Sign-up
(2) Login
(X) Exit
Enter: 1

Enter name: ays
Enter password: 1234567

Password must be at least 8 characters long.
Enter name: ays
Enter password: 12345678

(1) Sign-up
(2) Login
(X) Exit
Enter: 2

Enter name: ays
Enter password: 12345
The user either does not exist or password is incorrect. Try Again? (Y/N)
y

(1) Sign-up
(2) Login
(X) Exit
Enter: 2

Enter name: ays
Enter password: 12345678
Welcome
```

```
DAY: 1
Daily saved amount: 0
Weekly saved amount: 0

Enter today's budget: 100
Enter weekly goal amount: 200
budget overrun: 0
Today's adjusted budget: 100

(0) Cash Out
(1) End Day
(2) Configure
(3) Sign-out
(X) Exit
Enter: 0
Enter cash out amount: 20

========================================

DAY: 1
Daily saved amount: 80
Weekly saved amount: 0

budget overrun: 0
Today's adjusted budget: 100

(0) Cash Out
(1) End Day
(2) Configure
(3) Sign-out
(X) Exit
Enter: 1

========================================

DAY: 2
Daily saved amount: 0
Weekly saved amount: 80

Enter today's budget:
```

```
========================================

DAY: 2
Daily saved amount: 0
Weekly saved amount: 80

Enter today's budget: 100
budget overrun: 0
Today's adjusted budget: 100

(0) Cash Out
(1) End Day
(2) Configure
(3) Sign-out
(X) Exit
Enter: 2

========================================
(1) Change Budget
(2) Change Weekly Goal
(3) Savings Visibility: 2
(4) Go Back
(X) Exit
Enter: 3
Savings Visibility
(1) Disable     (2) Enable
Enter: 1
Savings Visiblity: Disabled
========================================
(1) Change Budget
(2) Change Weekly Goal
(3) Savings Visibility: 1
(4) Go Back
(X) Exit
Enter: 4

========================================

DAY: 2
Daily saved amount: *
Weekly saved amount: **************************
budget overrun: 0
Today's adjusted budget: ***
(0) Cash Out
(1) End Day
(2) Configure
(3) Sign-out
(X) Exit
Enter:
```

```
==========================================

DAY: 2
Daily saved amount: 0
Weekly saved amount: 80

budget overrun: 0
Today's adjusted budget: 100

(0) Cash Out
(1) End Day
(2) Configure
(3) Sign-out
(X) Exit
Enter: 0
Enter cash out amount: 110

==========================================

DAY: 2
Daily saved amount: 0
Weekly saved amount: 80

budget overrun: -10
Today's adjusted budget: 110

(0) Cash Out
(1) End Day
(2) Configure
(3) Sign-out
(X) Exit
Enter: 1

==========================================

DAY: 3
Daily saved amount: 0
Weekly saved amount: 80

Enter today's budget:
```

```
========================================

DAY: 3
Daily saved amount: 0
Weekly saved amount: 80

Enter today's budget: 100
budget overrun: -10
Today's adjusted budget: 90

(0) Cash Out
(1) End Day
(2) Configure
(3) Sign-out
(X) Exit
Enter: 3

(1) Sign-up
(2) Login
(X) Exit
Enter: 1

Enter name: haha
Enter password: 789456123

(1) Sign-up
(2) Login
(X) Exit
Enter: 2

Enter name: ays
Enter password: 1234567
The user either does not exist or password is incorrect. Try Again? (Y/N)
Y
Enter name: ays
Enter password: 12345678
Welcome
```

```
-----------------------------------------

DAY: 3
Daily saved amount: 0
Weekly saved amount: 80

budget overrun: -10
Today's adjusted budget: 90

(0) Cash Out
(1) End Day
(2) Configure
(3) Sign-out
(X) Exit
Enter: 1

=========================================

DAY: 4
Daily saved amount: 0
Weekly saved amount: 80

Enter today's budget:
```

```
==========================================

DAY: 7
Daily saved amount: 0
Weekly saved amount: 600

Enter today's budget: 100
budget overrun: 0
Today's adjusted budget: 100

(0) Cash Out
(1) End Day
(2) Configure
(3) Sign-out
(X) Exit
Enter: 0
Enter cash out amount: 0

==========================================

DAY: 7
Daily saved amount: 100
Weekly saved amount: 600

budget overrun: 0
Today's adjusted budget: 100

(0) Cash Out
(1) End Day
(2) Configure
(3) Sign-out
(X) Exit
Enter: 1
Congrats! You reached your goal amount!


==========================================
```

Conclusion

The results of this Budget Tracker program show that the system effectively helps the users to manage their daily expenses and it can develop financial discipline. By allowing users to set a daily budget, monitor cash outs, and track the weekly savings or goals, the program promotes awareness and accountability in their financial decision making. The feature that deducts overspending from the next day's budget teaches users to be more cautious with their spending habits, encouraging better financial control over time.

Overall, the project successfully meets its objectives by providing an easy to use and efficient tool for managing personal savings. It not only demonstrates practical applications of C++ programming but also highlights how technology can support financial literacy among students.

To continue enhancing the performance, reliability, and usability of the Budget Tracker Program, several enhancements are recommended. First, the implementation of data persistence is highly recommended. Integrating file handling or database storage would allow the program to save user credentials and financial data permanently, enabling users to log in and continue their progress in subsequent sessions. This improvement would greatly enhance the practicality and long-term usability of the program.

the development of a user interface with input validation, mechanisms to handle errors, and more intuitive and clear system prompts would make it even more user friendly. These features will not only overcome the functional limitations present in the current implementation but also create a more effective and reliable base for further use and development of the financial manager.

References

Hani, S., Nasyalia, C., & Maya, S. (2023). *The impact of financial literacy and financial inclusion towards the saving behavior of the students. MIX: Jurnal Ilmiah Manajemen*, 13(2). https://doi.org/10.22441/jurnal_mix.2023.v13i2.012

Nesia, I. L. N. H., & Sartika, F. (2022). *The influence of lifestyle and financial literacy on saving behaviour with self-control as a moderating variable: A study on young workers. International Journal of Economics Development Research*, 5(5), 4139–4156.

Torres, J., Geronimo, E., Estabas, E. M., Banquerigo, N., Buenaflor, E. F., Serapio, J. C., & Vidal, F. A. (2024). Mental accounting practices and spending behavior of collegiate students at National University Baliwag: An assessment. *SEISENSE Business Review, 4*(1), 181–199. https://journal.seisense.com/sbr/article/view/1087

Ablay, J. A. M. (2023). The relationship between spending behavior and student financial management skills. *IOER International Multidisciplinary Research Journal, 5*(2), 45–53.* https://www.ioer-imrj.com/wp-content/uploads/2023/06/The-Relationship-Between-Spending-Behavior-and-Student-Financial-Management-Skills.pdf