

**ADDIS ABABA SCIENCE AND TECHNOLOGY UNIVERSITY**

**COLLEGE OF ENGINEERING**

**DEPARTMENT OF SOFTWARE ENGINEERING (DEGREE PROGRAM)**

**Course Title:** Introduction to **Academic Year**: 2024/25

Software Engineering and Computing

**Course Code:** SWEG2101 **Section: A**

Group Members

1. Abigiya Fikru ETS0076/16

2.Arsema Seife ETS0192/16

3. Aster Teshome ETS0201/16

4. Barkot Frew ETS0222/16

3. Barkot Zerihun ETS0223/16

5. Beamlak Atilabachew ETS0229/16

**Submitted to:Kassahun A**

**Date of Submission: Monday January 5th , 2024**

**Table of Contents**

1. Introduction…………………...……………………………………………………………………………...……..…………1

1.1 Purpose and Goals……………….….………….……………………………………………………………….….…..1

1.2 Scope…………………….….………………………………………………….………………………….……….………….1

1.3 Target Audience………………………..…………………………………….………………………….……….………...2

2. System Requirements…………………………………………...………………………………………………….……..2

2.1 Hardware Requirements……………………………….……………………………………………………….…..….2

2.2 Software Requirements………………….……………………………………………………………..…………...2

3. System Design……………………….……………………………...……………………………..……………..….………2

3.1 Data Structure………………………………….………………………………………………………….……………...2

3.2Algorithm...................................................................………..................................................3

3.3 Flow Chart…………………………………………………….…………….……………..……..............................6

4. Program Implementation……………...………………...……………….………………….…..….…………....…..7

4.1 C++ code……………………………………………………………….………..…….….………….…………………….…7

4.2 Code organization and structure............................................……….....................................8

5. User Manual……………………………………………………………………………………..…….….....…….…….…..11

5.1 User Interface Guide.......................................................................……….......................…11

5.2 Troubleshooting…………………………………………………………....................….......…..................12

6. Conclusion…………………………………………………………………………………………….……………...…........12

6.1 Summary and Evaluation of the Project…………………………………………………….…………….......12

7. Github reference link.................................................................................………................…14

1. **Introduction**

**Project Overview**: **Academic Study Performance Analyser**

Second-year Software Engineering students prepare for their upcoming final exams and are keen on tracking their daily study efforts. They record their study hours for each day of the week in a notebook. At the end of the week, they aim to analyse their study patterns by calculating total weekly study hours and average daily study hours. To automate this analysis process, they seek a program that can efficiently handle their data.

**1.1 Purpose**

This project aims to develop a program that assists students in tracking and analysing their academic study performance. The program will enable students to input their daily study hours and generate a comprehensive report, including total weekly study hours, average daily study hours, and individual day breakdowns.

The primary goal of this project is to develop a C++ program that:

\* Reads and stores the names of students and their daily study hours.

\* Calculates the total study hours for each student for the week.

\* Determines the average study hours per day for each student.

\* Displays the results in a clear and organized tabular format.

\* Enables the user to search for specific students by name or ID.

\* Track Study Hours: Allow users to input their study hours for each day of the week.

\* Generate Report: Present the data in a clear and organized tabular format.

\* Search Functionality: Enable users to search for specific students by name or ID.

**1.2 Scope**

This project focuses on the following key aspects:

\* Data Input: Reading student names and their study hours for each day of the week.

\* Data Storage: Utilizing parallel arrays to store student information and study hours.

\* Data Processing: Calculating total weekly study hours and average daily study hours.

\* Data Output: Displaying results in a tabular format with headings.

\* Search Functionality: Implementing a search feature to locate specific students.

**1.3 Target Audience:**

\* Students: Primarily intended for students who want to monitor their study habits and improve their academic performance.

\* Teachers: Can be used by teachers to track student engagement and offer personalized guidance.

\* Researchers: May be useful for educational research to analyse study patterns and their impact on academic outcomes.

**2. System Requirements**

**2.1 Hardware Requirements**

\* Processor: Minimum Intel Core i3 or equivalent (dual-core recommended for better performance).

\* Memory (RAM): Minimum 4GB RAM (8GB recommended for smoother operation).

\* Storage: Minimum 100GB hard drive space

**2.2 Software Requirements**

\* Operating System: Windows 7 or later, macOS

\* Programming Language: Any suitable programming language like C++

\* C++ compiler (e.g., GCC, Clang, Visual Studio)

\* C++ IDE (e.g., Code::Blocks, Visual Studio Code, Eclipse CDT)

\* Internet Connection: for online resources, updates, and potential cloud-based storage.

**3. System Design**

**3.1 Data Structures**

\* *Parallel Arrays*:

\* studentNames: An array of strings to store student names/IDs.

\* studyHours: A 2D array of integers to store study hours for each student and each day.

\**Flow Control:*

\* Main Loop: Use a do-while loop to display the menu and repeatedly prompt for user input until the user chooses to exit the program.

\* Input Loop: Use a for loop to iterate through the number of students and prompt for their data.

\* Display Loop: Use a nested for loop to iterate through each student and each day of the week to display the study hours.

\* Search Loop: Use a for loop to iterate through the student names and compare them with the search name/ID.

\**Pointers:*

\* Pointers to Arrays: Use pointers to access and manipulate elements of the studentNames and studyHours arrays efficiently.

**3.2 Algorithm and Flowchart**

**Algorithm**:

1. Initialization:

\* Initialize constants: MAX\_STUDENTS, DAYS\_IN\_WEEK

\* Initialize arrays: students, studentCodes, studyHours, totals, averages

\* Initialize variables: numStudents, totalSum, overallAverage

2. Main Loop:

\* Display the welcome message and menu options based on numStudents

\* Read user’s choice

\* Process the choice using a switch statement:

\* Case 1: Add Student

\* Read student name and code

\* For each day of the week:

\* Read study hours (validate input)

\* Update studyHours array

\* Calculate total study hours for the student and update totals array

\* Calculate average study hours for the student and update averages array

\* Update totalSum and numStudents

\* Case 2: See Results

\* If numStudents is 0, display a message

\* Otherwise:

\* Calculate and display overall total study hours

\* Calculate and display overall average study hours

\* Display a table with student names, codes, daily study hours, total hours, and average hours

Case 3: Search Student

\* If numStudents is 0, display a message

\* Otherwise:

\* Read student name or code to search by the choice of the user input

\* Iterate through the students:

\* If the name and code match, display detailed information about the student

\* If no match is found, display a “not found” message

\* Case 4: Exit

\* Display a goodbye message and terminate the program

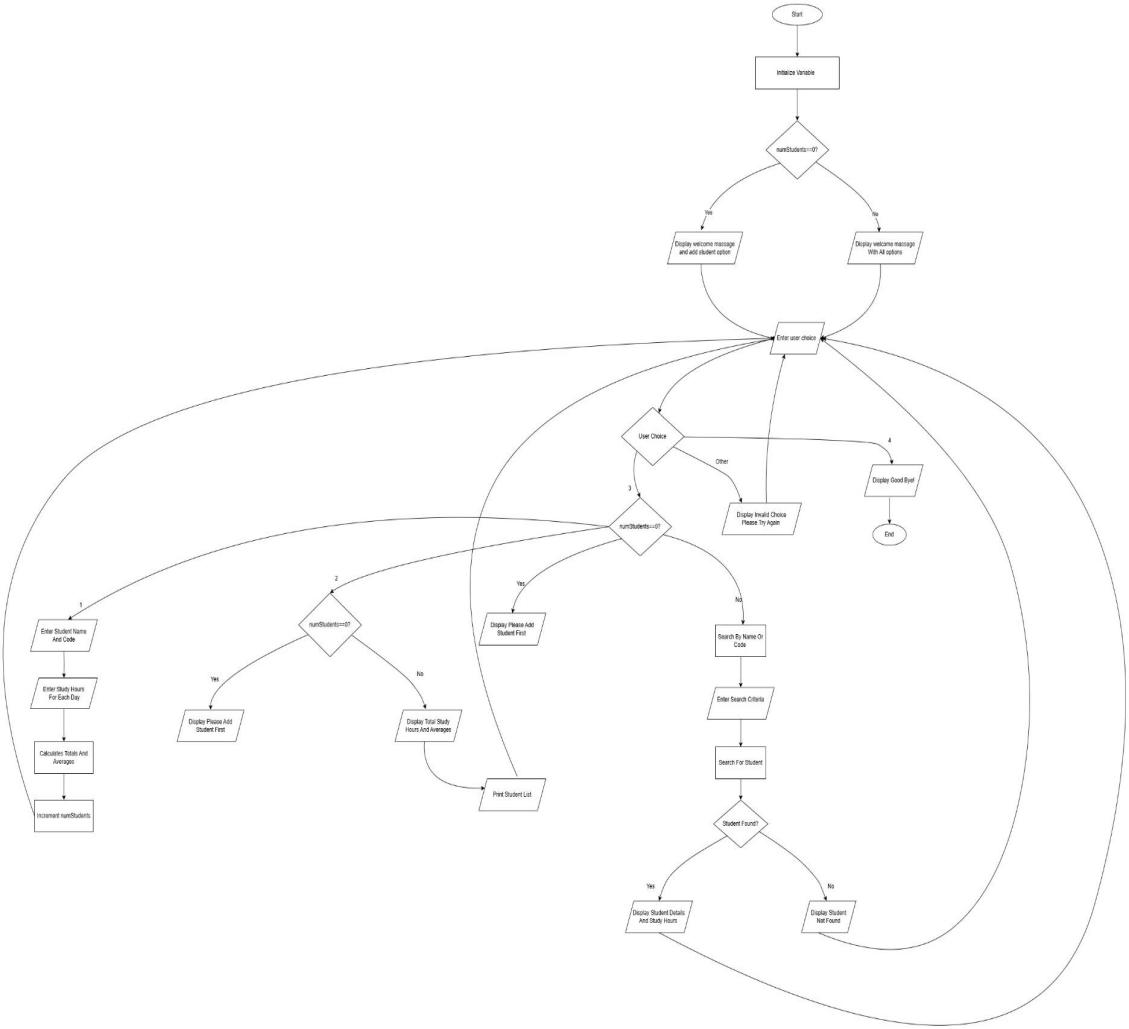
\* Default: Invalid Choice

\* Display an error message

3. End of Loop:

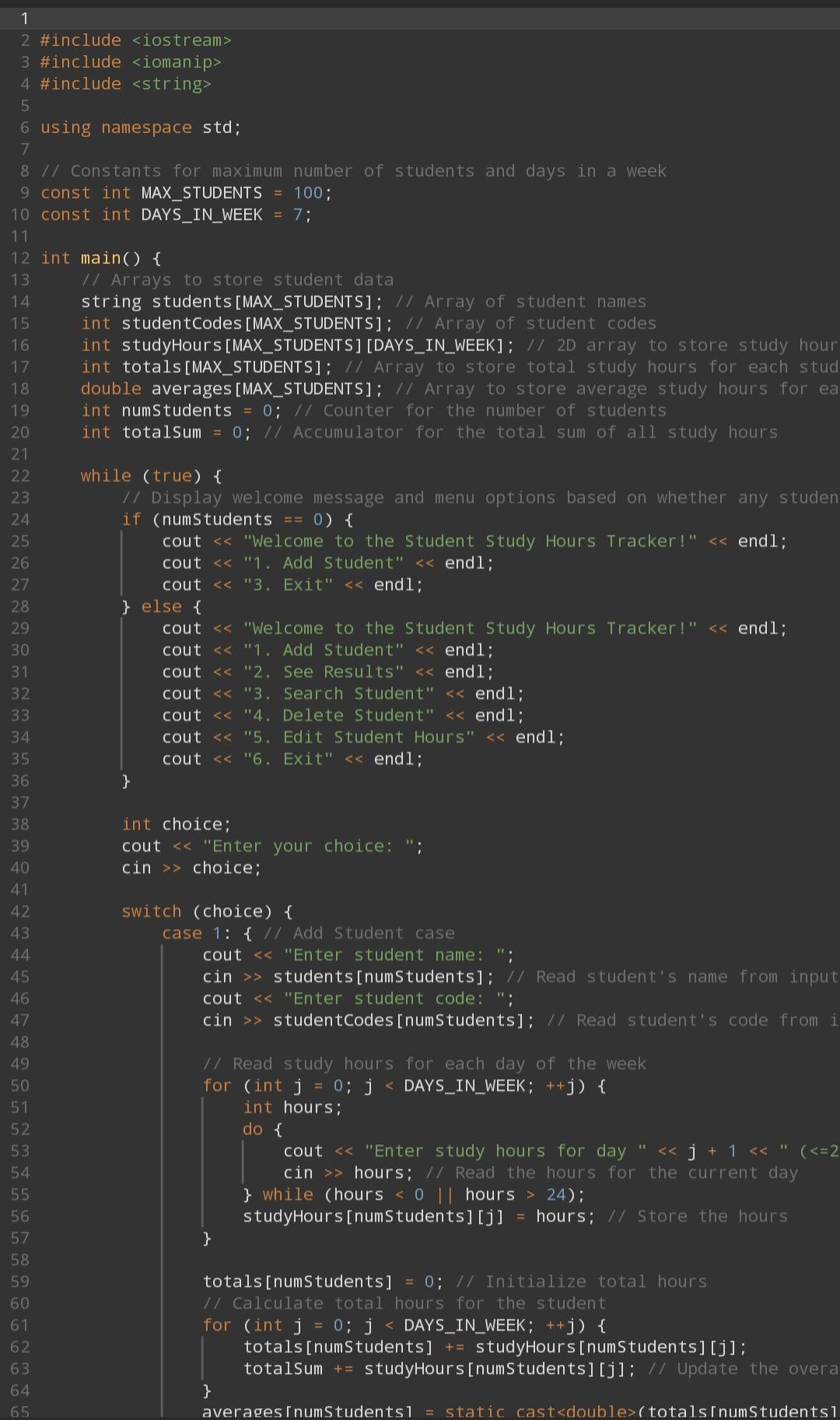
**-** Repeat the main loop until the user chooses to exit

**Flow Chart**

****

**4. Program Implementation**

4.1 C++ code



**4.2 Code organization and structure**

Lines 1-2: Includes the necessary libraries. `<iostream>` is for input and output operations, and `<iomanip>` is for formatting (like setting)

Line 3: Enables the use of standard library features without needing to prefix them with `std::’.

Lines 4-5: Defines constants. `MAX\_STUDENTS` sets the maximum number of students to 100, and `DAYS\_IN\_WEEK` is set to 7 for the days of the week.

Line 6 : Starts the main function, which is the entry point of the program.

Lines 7-11: Declares arrays for storing student names, student codes, study hours for each day of the week, total study hours, and average study hours for each student.

Lines 12-13: Initializes `numStudents` to 0 (indicating no students have been added) and `totalSum` to 0 (for total study hours).

Line 14: Begins an infinite loop to repeatedly show the menu until the user chooses to exit.

Line 15: Checks if no students have been added to decide which menu to display.

Lines 16-18: Displays the welcome message and menu options if there are no students (only options to add a student or exit).

Line 19: If there are students, this part of the code executes.

Lines 20-26: Displays a more detailed menu with options to add a student, see results, search for a student, delete a student, edit hours, or exit.

Line 27: Ends the `if-else` statement.

Lines 28-30: Declares an integer variable `choice` to store the user’s menu selection, prompts for the choice, and reads it from input.

Line 31: Begins a switch statement to handle the user's menu choice.

Line 32: Begins the case for adding a student.

Lines 33-36: Prompts for and reads the name and code of a new student, storing them in the respective arrays.

Line 37: Begins a loop to gather study hours for each day of the week.

Lines 38-42: Declares a variable `hours` and prompts the user to enter study hours for each day. It ensures that the input is between 0 and 24 using a do-while loop.

Line 43: Stores the valid hours entered for each day into the `studyHours` array for the current student.

Line 44: Ends the loop for entering study hours.

Line 45: Initializes the total study hours for the new student.

Line 46: Begins another loop to calculate the total study hours for the student.

Lines 47-48: Accumulates the total study hours for the student and adds them to the overall total (`totalSum`).

Line 49: Ends the loop for calculating totals.

Lines 50-53: Calculates the average study hours for the student, increments the total student count, and breaks out of the switch statement.

Line 54: Begins the case for viewing results.

Lines 55-58: Checks if there are any students to show results for; if not, prompts the user to add students.

Lines 59-60: Displays the total study hours for all students and the overall average study hours.

Lines 61-63: Prints a formatted table header for the results.

Line 64: Starts a loop to print the results for each student.

Lines 65-68: Prints the student's name, code, and study hours for each day in a formatted manner.

Lines 69-70: Prints the total and average study hours for each student and ends the loop.

Lines 71-73: Prints a closing line for the table and breaks out of the switch statement.

Line 74: Begins the case for searching a student.

Lines 75-78: Checks if there are students to search for; if not, prompts to add students.

Lines 79-84: Declares variables to store search criteria and prompts the user to choose whether to search by name or code.

Lines 85-91: Reads either the student's name or code based on the user's choice.

Lines 92-94: Initializes a `found` flag and loops through the students to find a match based on the search criteria.

Lines 95-97: If a match is found, prints the student's details, including total and average study hours.

Lines 98-100: Prints headers for the study hours per day.

Lines 101-103: Loops through each day and prints the study hours for that student.

Lines 104-108: Closes the day hours table, marks the student as found, and breaks out of the loop.

Lines 109-111: If no student was found after the loop, it informs the user.

Lines 112-116: Begins the case for deleting a student, checking if there are any students to delete; if not, it displays a message.

Lines 117-119: Prompts for the code of the student to delete.

Lines 120-123: Initializes a `found` flag and loops through the students to find the student by code.

Lines 124-125: If found, adjusts the `totalSum` by subtracting the total hours of the student to be deleted.

Lines 126-135: If found, shifts all subsequent student data left to fill the gap created by the deletion.

Lines 136-140: Decrements the student count, confirms successful deletion, and breaks out of the loop.

Lines 141-143: If the student wasn't found, it informs the user.

Lines 144-148: Begins the case for editing student hours, checking if there are students; if not, displays a message.

Lines 149-151: Prompts for the code of the student to edit.

Lines 152-155: Initializes a `found` flag and loops through the students to find the student by code.

Lines 156-159: If found, confirms the student being edited and adjusts the `totalSum`, resetting the total hours for that student.

Lines 160-165: Loops through each day to prompt for new study hours, ensuring they are valid.

Lines 166-169: Updates the study hours for the student, recalculates their total, and updates the overall total.

Lines 170-174: Calculates the new average study hours, confirms the update, and breaks out of the loop.

Lines 175-177: If the student wasn't found, informs the user.

Lines 178-180: Prints a goodbye message and exits the program.

Lines 181-182: Handles any invalid menu choices by prompting the user to try again.

Lines 183-185: Ends the switch statement, the infinite loop.

**5. User Manual**

**5.1 User Interface Guide and Troubleshooting**

A brief overview of the user interface and how to interact with it:-

*Main Menu*:

- The main menu is displayed on the console.

**-**  You can select options by entering the corresponding number and pressing Enter.

**-** The program will prompt you for input when necessary.

*Input Screen:*

**-** The input screen will ask you to enter the number of students and the study hours for each student.

**-** Enter the required information and press Enter.

*Display Screen:*

**-** The display screen will show a table with student names/IDs, daily study hours, total study hours, and average study hours.

*Search Screen:*

**-** The search screen will prompt you to enter the name or ID of the student you want to search for.

**-** Enter the name or ID and press Enter.

**5.2 Troubleshooting**

If you encounter any issues while using the program, please refer to the following tips:

***-*** *Invalid Input:* Ensure you enter valid numerical values for study hours.

**-**  *Program Crashes:* If the program crashes unexpectedly, try running it again or contact the program developer.

**-** *Incorrect Output:* Double-check your input data and the program's calculations. If you find any discrepancies, contact the program developer.

**6. Conclusion**

**6.1 Summary and Evaluation**

The "Academic Study Performance Analyzer" aims to assist software engineering students in tracking and analyzing their study habits. The program successfully meets its objectives by utilizing parallel arrays to store students’ names and their daily study hours, implementing functions to calculate and display total study hours for each student for the week, average study hours per day across all students presenting the output in a tabular format as shown in the example: including a search feature that enables users to retrieve details for a specific student.

**Benefits of the program:**

**Time-Saving**: Automates the process of tracking and analysing study hours.

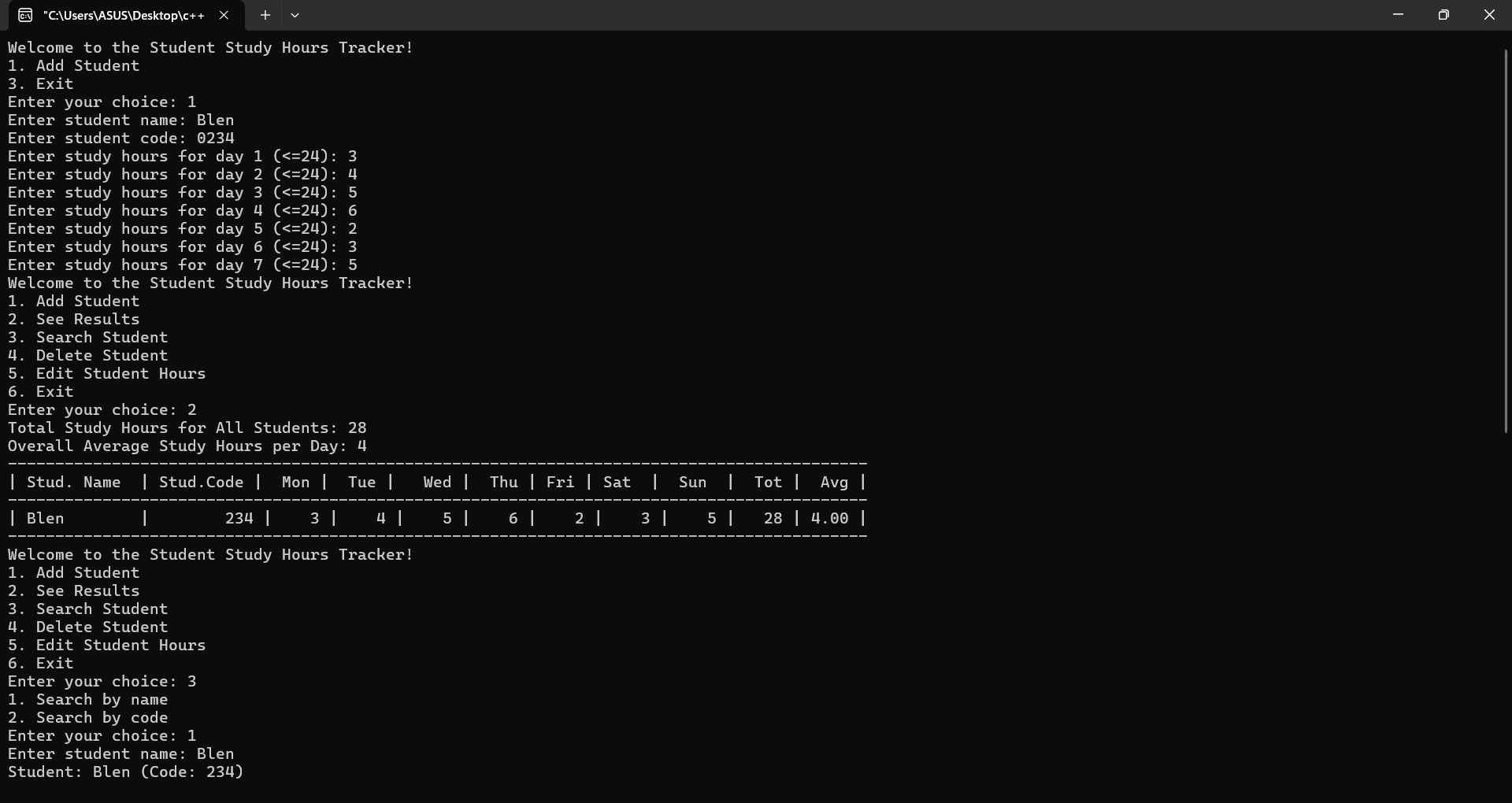
**Improved Performance Analysis**: It helps students identify study patterns and adjust as needed.

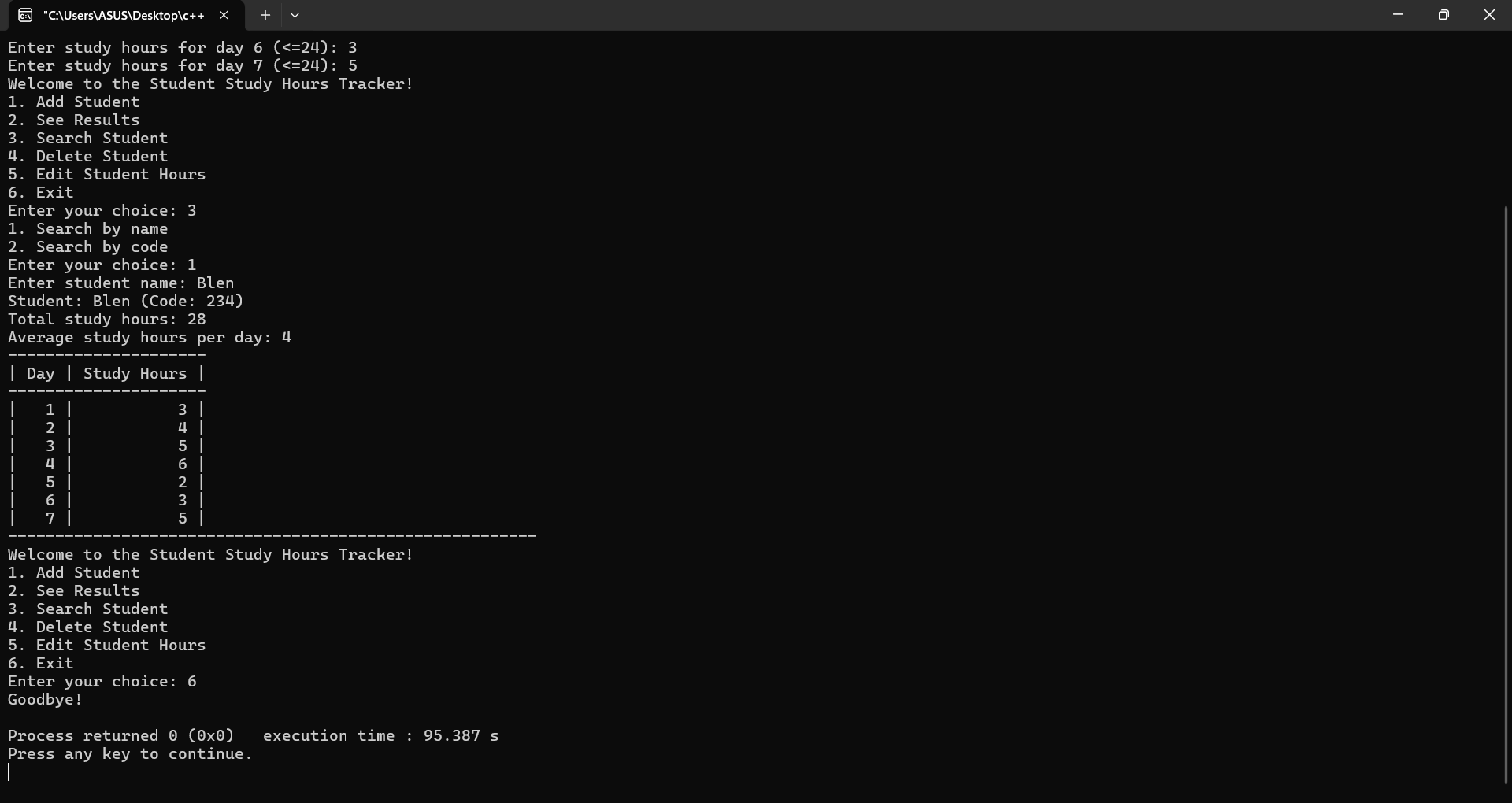
**Customizable and Scalable**: Adaptable for any number of students and can be extended to include additional features.

**User-Friendly Interface:** The clear tabular representation makes it easy for students and instructors to interpret the data.

This program provides a practical tool for students aiming to optimize their study schedules and prepare more effectively for exams.

Output result





**7.GitHub reference link**