STUDENT AND TEACHER TASK SCHEDULER

Braganza, Ralph Angelov F.
Condino, Niel Vincent B.
Lisud, Ian Paulo S
Lopez, Andrei Dion C

Technological Institute of the Philippines
Quezon City

November 2025

Table of Contents

Introduction

Students often experience stress and decreased performance when being overloaded with heavy workloads and simultaneous deadlines. When multiple assignments are due at the same time, students might struggle to balance these said tasks. Such overlap often results in decreased assignment quality and unhealthy effects on student well-being. Research has also shown that students often feel academic pressure due to heavy loads of work, which can be stressful when handling many difficult assignments from different classes all at once. This can significantly affect the student's physical health, which leads them to lose interest in eating regularly due to multiple tasks, develop constant headaches, and experience severe fatigue (Vallejo, 2023). According to Zhang et al. (2022), students exposed to a high academic stress environment may experience anxiety, and this anxiety may further contribute to the occurrence of depression and hopelessness in their academic journey. Teachers may sometimes be unaware of their students' overall workload when setting deadlines. This lack of coordination contributes to the increased stress levels and diminished academic performance when passing an assignment. Therefore, implementing a scheduling system that allows teachers to monitor section-wide deadlines could minimize conflicts, improve workload balance for students and enhance overall communication between teachers and students.

Currently, teachers only have limited visibility on the overall workload assigned to a particular section. This makes it hard for them to set fair and manageable deadlines for new tasks. As a result, students often face multiple tasks that have the same submission dates, which can cause stress and reduce their quality of work. According to Gordon (2023), effective task performance requires the elimination of distractions and dividing assignments into manageable parts to stay organized, stay focused, and avoid feeling overwhelmed. Task student-teacher deadline management enables monitoring workload distribution which prevents deadline overload, and supports teachers in establishing reasonable timelines that enhance the student's output quality.

The purpose of this system is to assist both teachers and students in managing overlapping assignment workloads across different subjects. The program allows teachers to gain better awareness by providing insights into students' deadlines. This ensures that teachers can set reasonable deadlines for each task to prevent rushed, low-quality, or late submissions. This system could possibly reduce academic stress experienced by students and help them track deadlines and organize their time effectively.

The Project

The project aims to bridge the gap between the teachers and the students about their deadlines being assigned on the same day with other tasks. The program allows teachers to view students' tasks through a calendar, in which the teachers are able to see the tasks assigned by them and other teachers. This enables teachers to visualize overlapping deadlines across subjects, thereby reducing the likelihood of conflicting task schedules. The program allows students and teachers to register, where the program will ask for their username, password, and what they want to be logged in as. Once logged in using their credentials, which are stored within the system's internal data structure, they are then prompted to a menu.

If the user is logged in as a teacher, they are prompted to the teacher's menu, where they are able to use their arrow keys to choose the following options:

The **Add a task** option is where the user will be able to add a task. First, the program will ask the user to specify a date to assign the task, and then they have the option to add a maximum of 10 tasks per day. The user is then asked to input a title and what time of day the task will be available and unavailable.

The **View task** option will let the user view the task that is on a specific date by asking for the date of when the task was assigned. Once the inputted date is specified, it will then display the available tasks that were assigned on that day either by the user or by other teachers.

The **Delete task** option will allow the user to delete tasks and will also have the option to delete certain tasks instead of everything.

The **View Calendar** option lets the user view a simple calendar to see the highlighted dates representing tasks assigned by the user or other teachers.

The **Logout** option will let the user log out from their account.

If the user is logged in as a student, they are prompted to the student's menu, where they are able to use their arrow keys to choose the following options:

The **View task** option will let the user view the task on a specific date by asking the user for the date of when the task was assigned. Once the inputted date is specified, it will then display the available tasks that were assigned on that day by their teachers.

The **View Calendar** option lets the user view a simple calendar to see the highlighted dates of tasks assigned.

The **Logout** option will let the user log out from their account.

Despite its simplicity, the system demonstrates the potential to improve coordination of assignment scheduling among teachers and students by giving them an easier way to see tasks.

Objectives

General Objective:

- To develop a task scheduling system that helps both teachers and students track, organize and manage tasks efficiently.

Specific Objectives:

- Create a login and registration system for both teachers and students.
- To allow teachers to:
    - Display the amount of tasks given in a day from all the teachers in a section to know if they overlap other subjects.
    - Set tasks and delete tasks for students
- To allow students to:
    - Display the amount of tasks given in a day from all the teachers in their section
- Test and evaluate the system's accuracy.

Flowchart of the System

This section presents the flowchart representation of the different processes of the program. It provides a visual overview of the program's logical flow, illustrating how different processes occur within the system. It visually maps out the sequence of action, making it easier to comprehend the interactions between different components of the code The flowcharts primarily focus on the logic behind the program's create, read, update, and delete functions, offering a clearer understanding of the relationships between the components of the code.

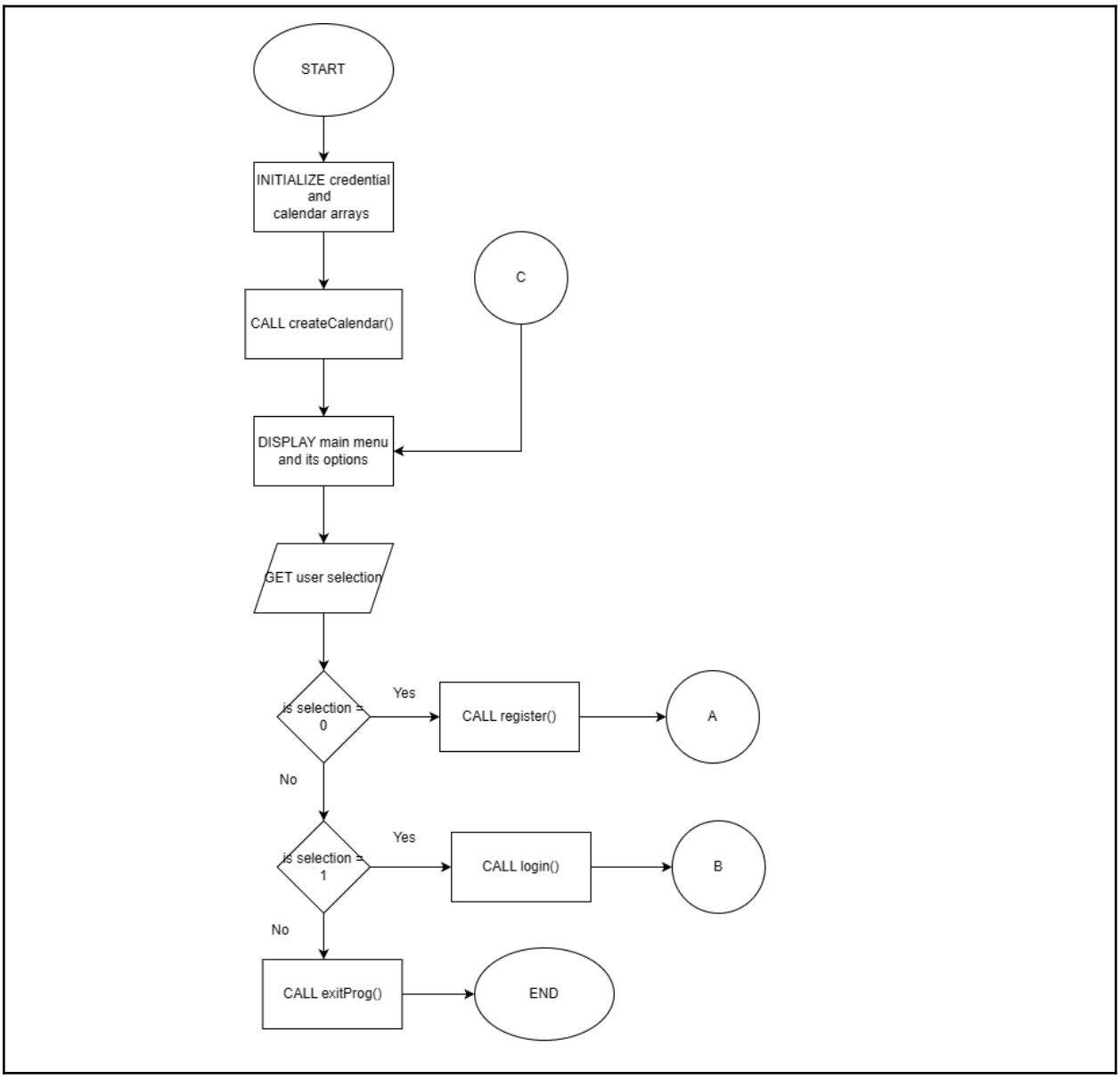


Figure 1: Main Menu Logic

Figure 1 illustrates the process of the Student and Teacher Task Scheduler program's main menu. It begins by initializing the credential and calendar arrays, then calls the createCalendar() function to build the calendar structure for the system. After setup, the program displays the main menu and prompts the user to choose between registering, logging in, or exiting.

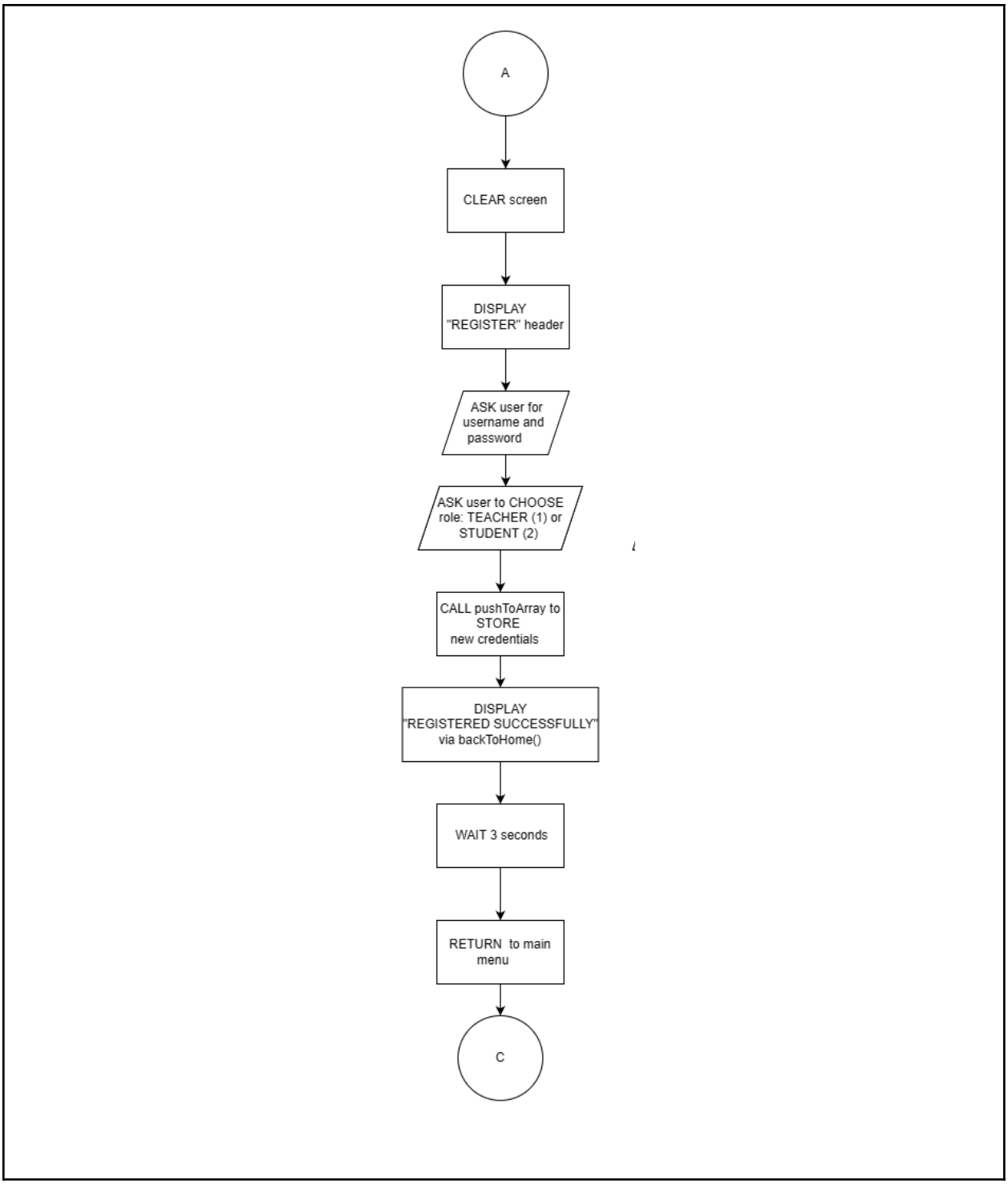


Figure 2: Registration Logic

Figure 2 illustrates the process of the program's registration logic. From figure 1, if the user selects registration, the system clears the screen, asks for a username, password, and role, and stores the new credentials using pushToArray(). Once registration is successful, a confirmation message is displayed, a short delay occurs, and the program returns to the main menu.

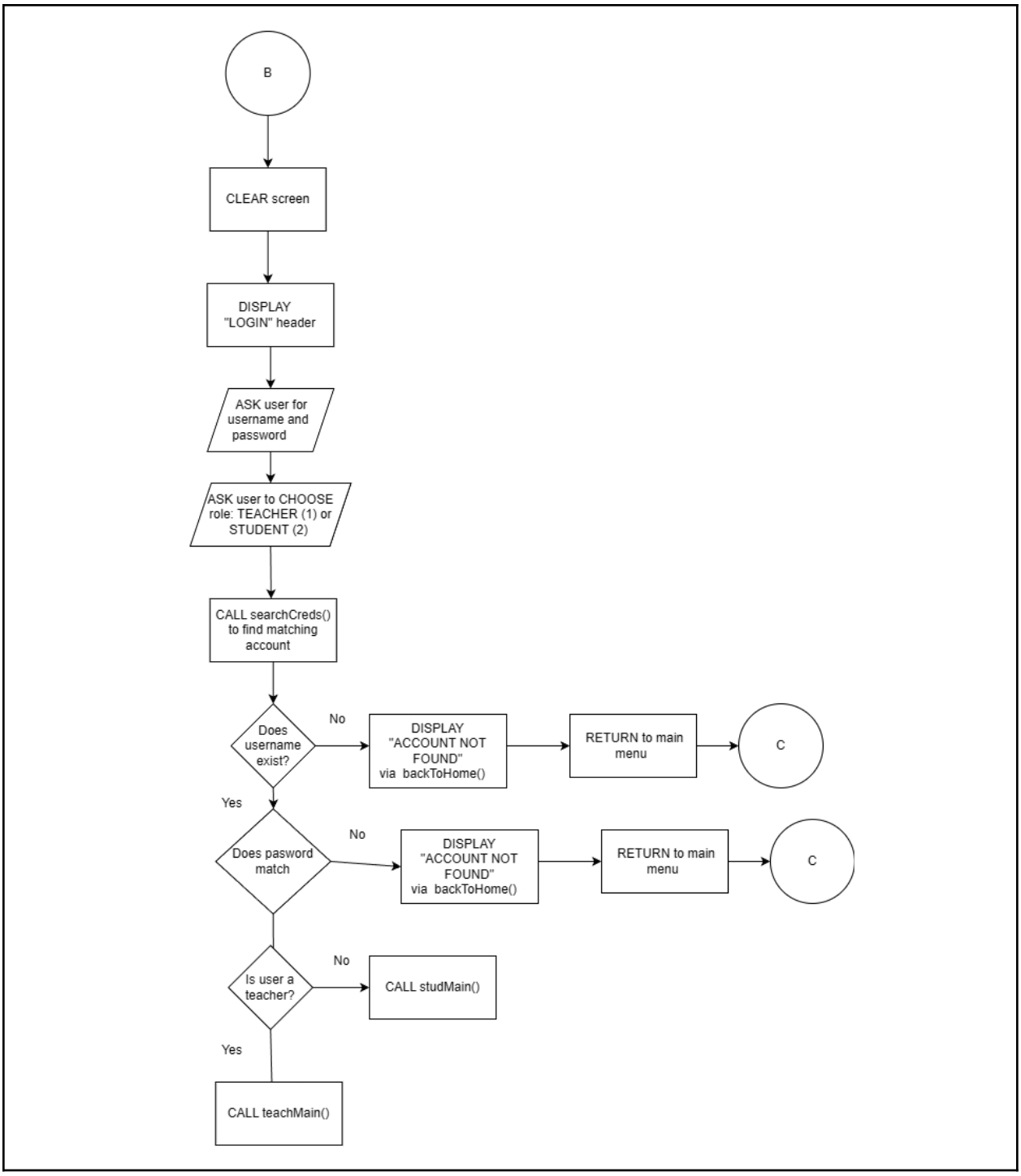


Figure 3: Login Logic

Figure 3 illustrates the login process of the program. From figure 1, if the user chooses to log in, the system requests credentials and a role, searches for a matching account, and verifies the username and password. Depending on the result, the program either directs the user to the appropriate main interface (teachMain() or studMain()) or displays an error message and returns to the main menu.

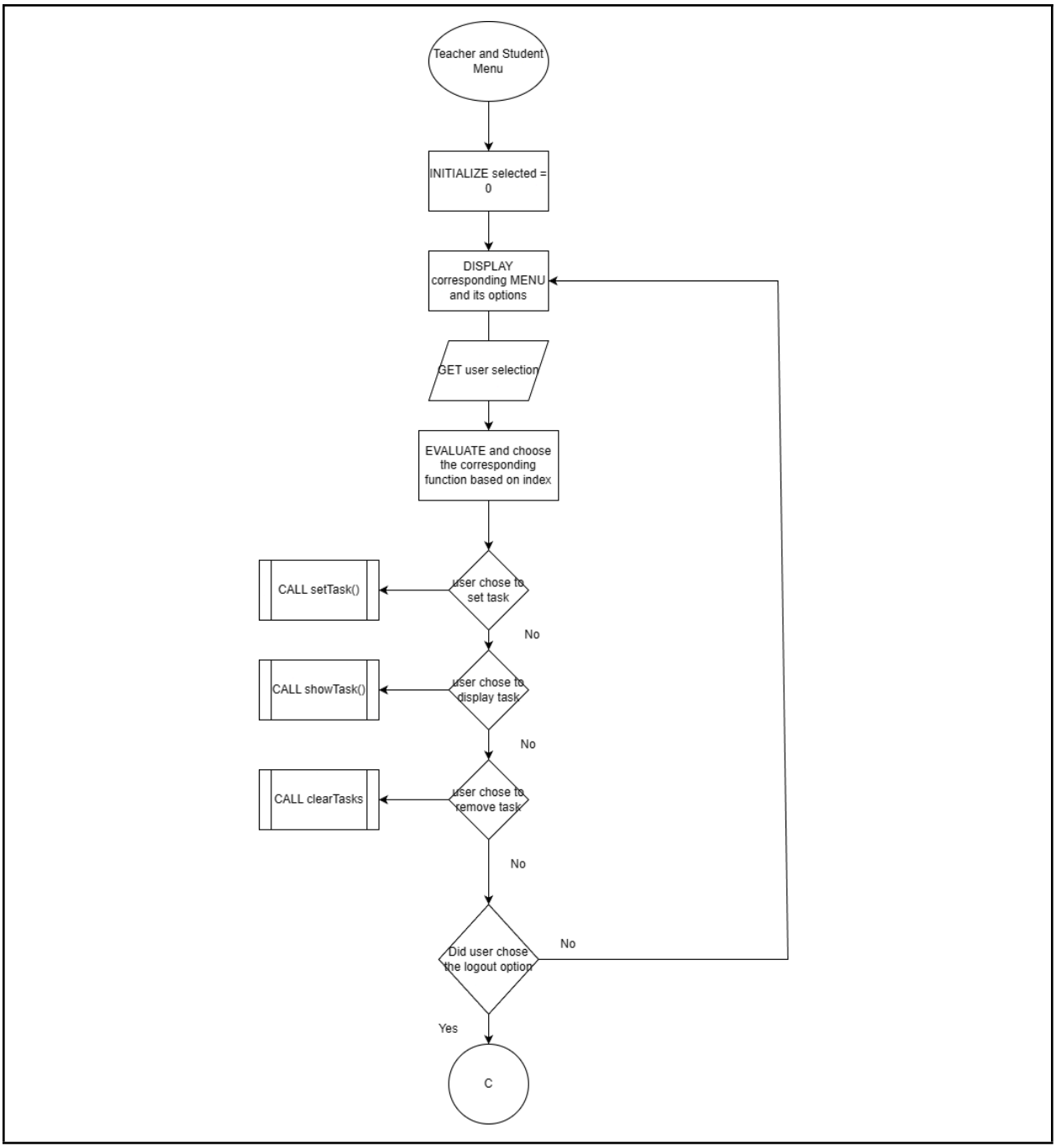


Figure 4: Teacher and Student Menu Logic

Figure 4 illustrates the logic of the teacher and student menu system. It begins by initializing a variable named selected to zero and displaying a menu with several options. The user then selects an option, and the program evaluates the selection to determine which function to execute. Depending on the choice, the program may call setTask() to add a task, showTask() to display existing tasks, or clearTasks() to remove tasks. After executing the chosen function, the program checks if the user selected the logout option. If not, it loops back to display the menu again; otherwise, it goes back to the main menu as shown in figure 1.

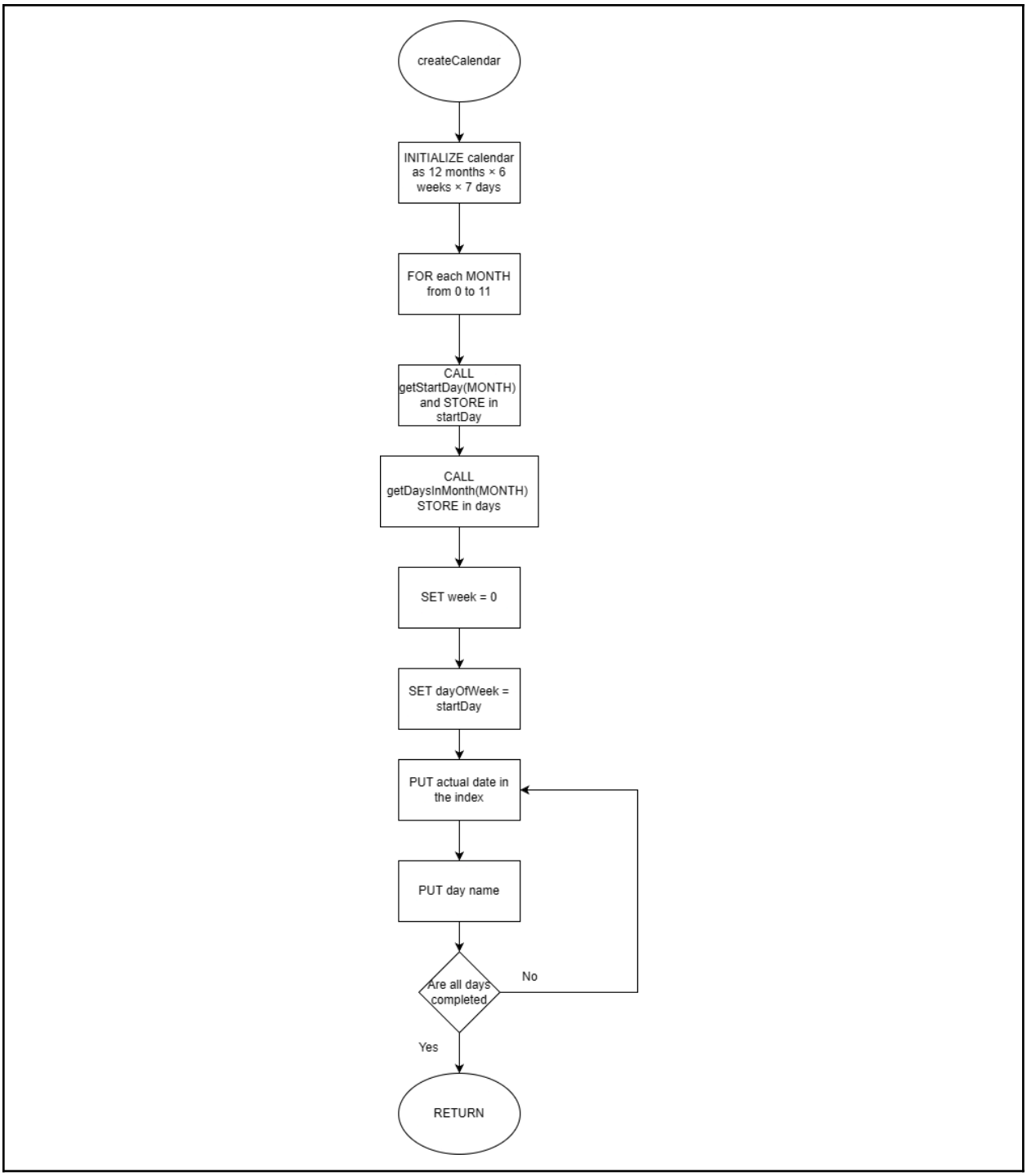


Figure 5: Calendar array formatting Logic

Figure 5 illustrates how the createCalendar() function formats a calendar structure. It starts by initializing a 12x6x7 calendar array representing months, weeks, and days. For each month, the program calculates the starting weekday using getStartDay() and determines the number of days using getDaysInMonth(). It then loops through each day of the month, storing the date and corresponding day name in the calendar grid. Finally, the process repeats for all twelve months, completing the full year's calendar before ending.

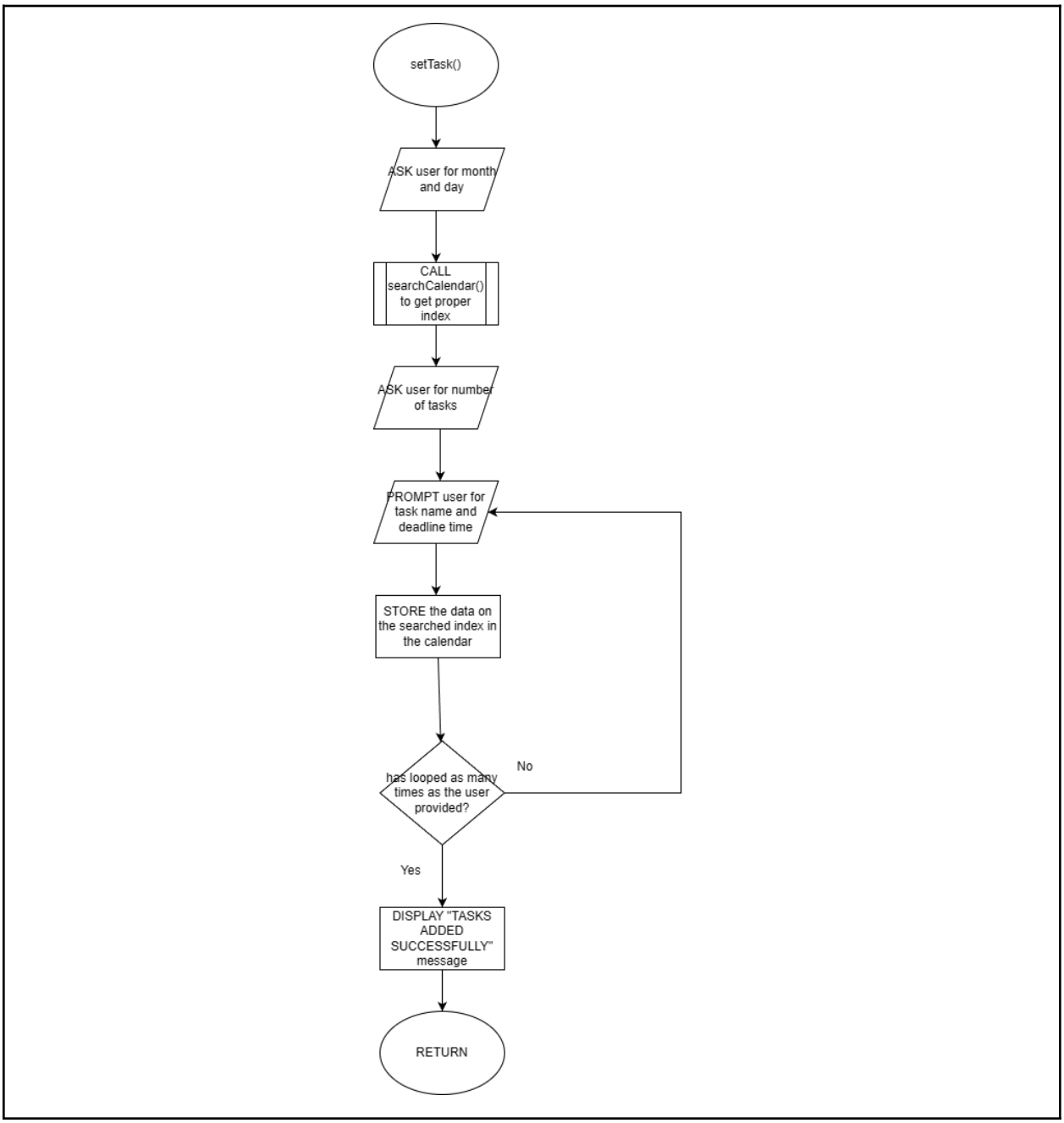


Figure 6: Storing task Logic

Figure 6 illustrates how the code allows teachers to add tasks to specific calendar dates. It begins by asking the user to input a month and day, then uses searchCalendar() to locate the correct date index in the calendar. The program asks for the number of tasks to add and loops that many times to collect each task's name and deadline. Each entered task is stored in the corresponding date slot in the calendar. Finally, the system confirms completion by displaying a "Tasks Added Successfully" message before ending the process.

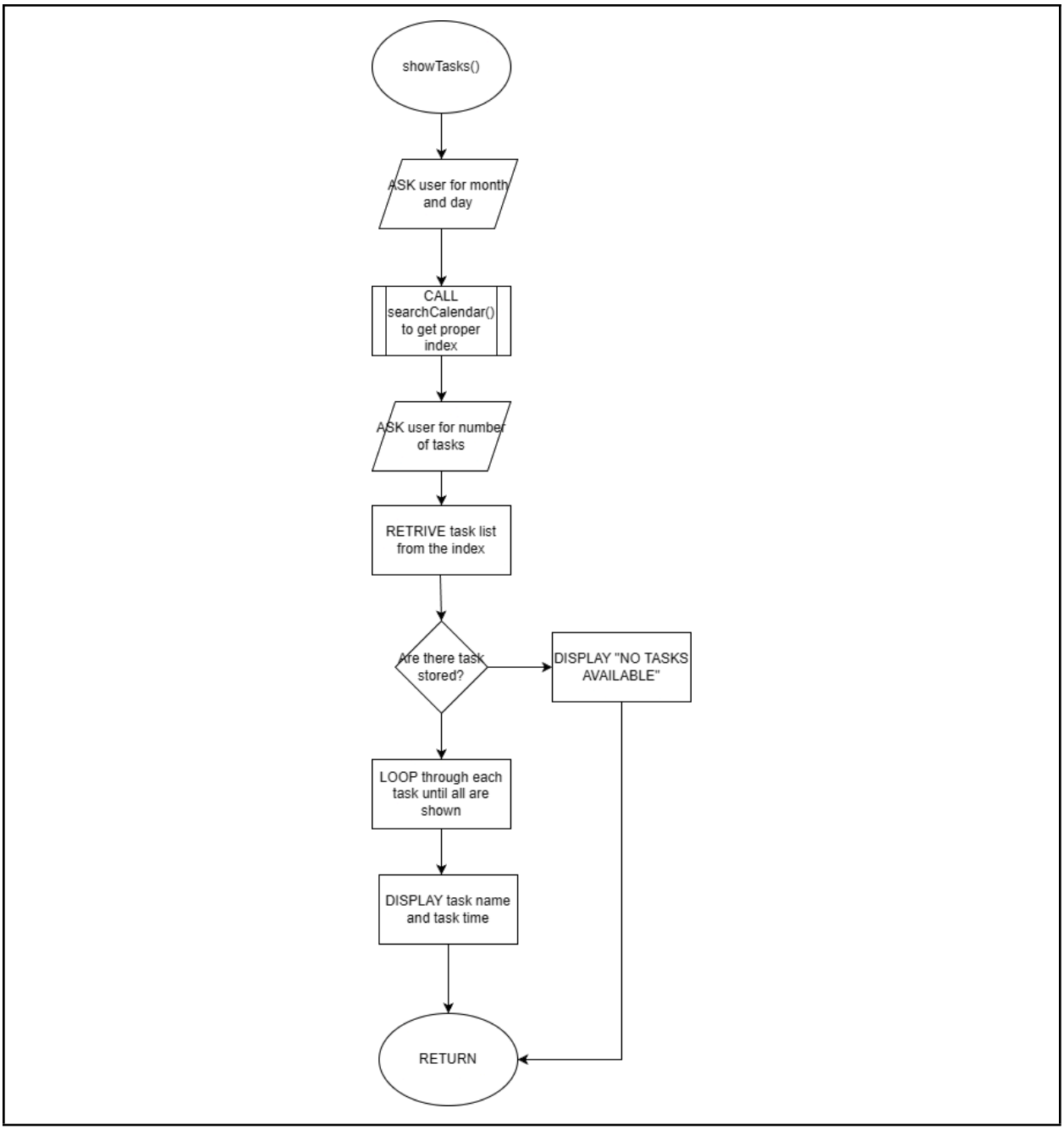


Figure 7: Displaying task Logic

Figure 7 illustrates a process for displaying scheduled tasks from a calendar system. It begins by asking the user for a specific month and day, then calls the searchCalendar() function to locate the correct index in the calendar. After that, the system retrieves the list of tasks associated with that date. If no tasks are found, it displays “NO TASKS AVAILABLE.” Otherwise, it loops through each stored task and displays the task name and time until all tasks are shown.

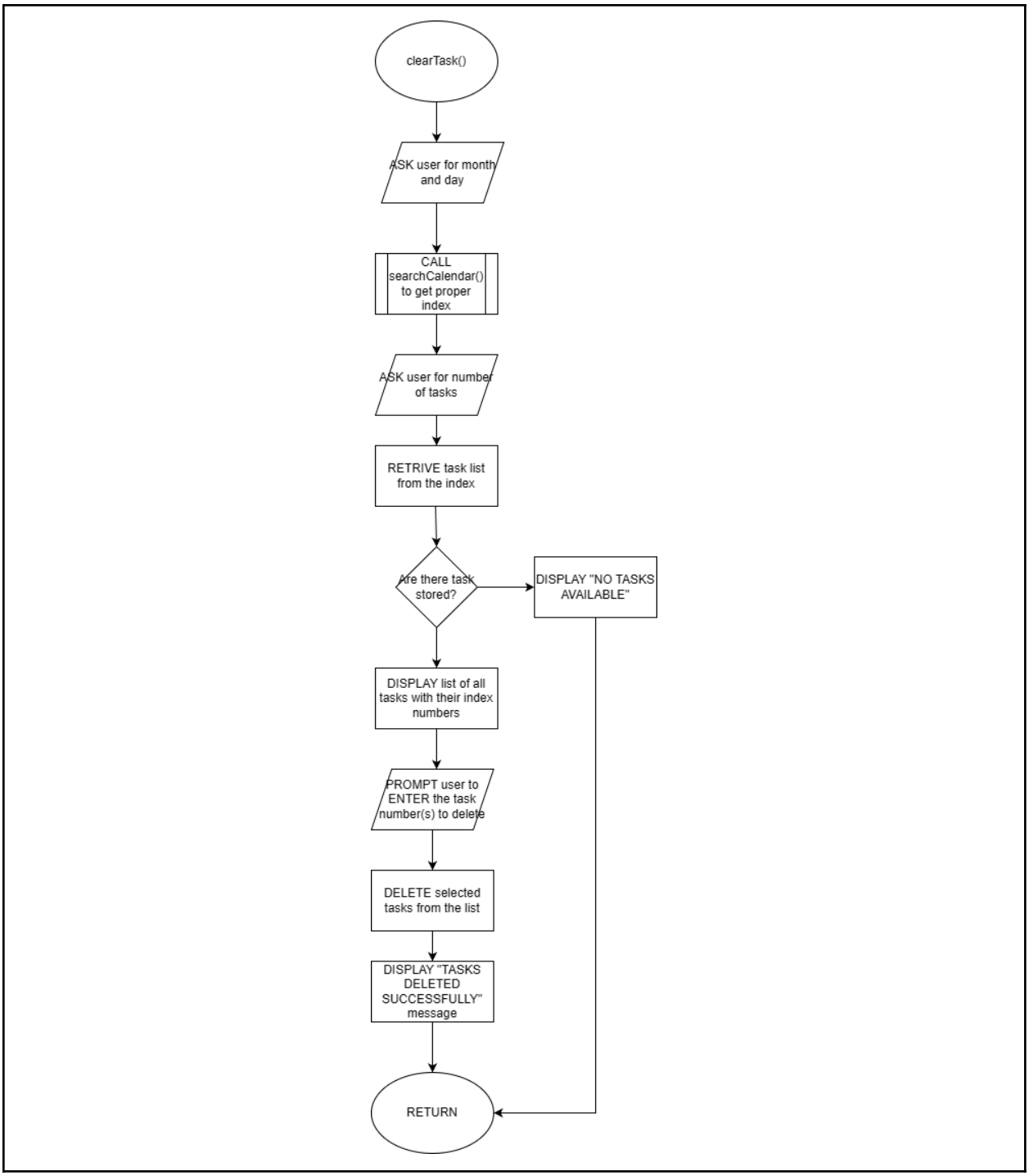


Figure 8: Deleting task Logic

Figure 8 illustrates the process of how the deleting task in the code works. It starts by asking the user for the month and day, then uses searchCalendar() to find the correct index in the calendar. The program retrieves the list of tasks for that date and checks if any exist. If no tasks are found, it displays a “No Tasks Available” message; otherwise, it shows the list of tasks with their index numbers. The user selects which tasks to delete, and the program removes them, confirming the action with a “Tasks Deleted Successfully” message before ending.

Pseudocode

This section presents the pseudocode of the program which outlines the logical flow and main procedures without using specific programming syntax. Pseudocode provides a clear understanding of how the program operates, making it easier to analyze the code before the actual implementation. It highlights the logic, decision-making processes, input and output handling, and any important computations or data manipulations performed in the program.

calendar.cpp

```
START
     INITIALIZE calendar for year 2025

     FUNCTION createCalendar(calendar)
          CREATE calendar for month and day

     FUNCTION printCalendar (schedINfo)
          PRINT the calendar

     FUNCTION validDate(month, day)OF Integer
          Checks for valid month and day

     SET schedInfo searchCalendarDate(calendar, message)
          INITIALIZE month and day OF Integer
          INITIALIZE isDateValid OF Boolean
              PRINT message
          INPUT month and day
          IF isDateValid not equal to validDate
              PRINTS "Invalid date\n"

     FUNCTION setTask (calendar)
          INITIALIZE countTask
          PRINT "Select month and day for deadline:\n"
```

```
          PRINT "Enter task amount:  "
          INPUT countTask from 1 to 10
          PRINT "Enter task name"
          INPUT taskName
          PRINT "Enter deadline time: "
          INPUT deadlineTIme
     Until loop is done
     PRINT "Task successfully Added"

     FUNCTION showTasks()
          PRINT "Show tasks from the date...\n"
          INPUT month and date
          PRINT "Available tasks"
          IF date is empty
          PRINT "  No tasks scheduled.\n"
          PRINT Deadline Time

     FUNCTION clearTasks ()
          PRINT "Select task to be deleted\n"
          INPUT month day
          IF date is empty
          PRINT "  No tasks scheduled.\n"
          PRINT "To be Deleted"
          INPUT selection
          PRINT "Deleted selected tasks.\n"
STOP
```

Figure 9: Pseudo code of the calendar.cpp file

This figure represents how the calendar.cpp works. It shows the calendar creation, day verification, analysis if the year is a leap year, searching of specific contents in a user inputted date, and the creation, deletion, and displaying of tasks in the calendar.

credentials.cpp

```
START
    FUNCTION backToHome(message):
        PRINT message
        WAIT 3 seconds
        CALL opening()

    FUNCTION searchCredentials(username, password, role):
        IF username IS EMPTY OR password IS EMPTY THEN
            CALL backToHome("Missing fields")
        END IF
        found = false
        FOR i = 0 TO credSize - 1:
            IF role == 2 AND studentCredentials[i].username ==
            username
                found = true
                IF studentCredentials[i].password == password
                THEN
                    CALL studentMain()
                ELSE
                    CALL backToHome("Wrong username or
                    password")
                END IF
                BREAK
                ELSE IF role == 1 AND
                teacherCredentials[i].username == username
                found = true
                IF teacherCredentials[i].password == password
                THEN
                    CALL teacherMain()
```

```
            ELSE
                CALL backToHome("Wrong username or
                password")
            END IF
            BREAK
        END IF
    END FOR

    IF not found
        CALL backToHome("Account not found")
    END IF

FUNCTION login():
    CLEAR SCREEN
    CALL printHeader("LOGIN")
    PRINT "Username: "
    INPUT username
    PRINT "Password: "
    INPUT password
    CALL isNumInRange("Login as teacher(1) or student(2):
    ", role, 1, 2)
    CALL searchCredentials(username, password, role)

FUNCTION pushToArray(username, password, role):
    IF username IS EMPTY OR password IS EMPTY THEN
        CALL backToHome("Empty credentials")
    END IF

    FOR i = 0 TO credSize - 1:
        IF role == 2 THEN
            target = studentCredentials[i]
```

```
            ELSE
                target = teacherCredentials[i]
            END IF

            IF target.username == username THEN
                CALL backToHome("Username already exists")
            END IF

            IF target.username IS EMPTY THEN
                target.username = username
                target.password = password
                BREAK
            END IF
        END FOR

    FUNCTION registerCredentials():
        CLEAR SCREEN
        CALL printHeader("REGISTER")
        PRINT "Username: "
        INPUT username
        PRINT "Password: "
        INPUT password
        CALL isNumInRange("Register as teacher(1) or
        student(2): ", role, 1, 2)
        CALL pushToArray(username, password, role)
        CALL backToHome("Registered Successfully!")

    FUNCTION opening():
        CALL printMenu("School Task Scheduler", openMenuItems,
        openMenuSize)
        selected = selection(openMenuItems, printOpeningMenu)
```

```
        SWITCH selected:
            CASE 0: CALL registerCredentials()
            CASE 1: CALL login()
            CASE 2: CALL exitProg()
STOP
```

Figure 10: Pseudo code of the credentials.cpp file

Figure 10 shows the workflow of the credentials.cpp file. It shows the log-in and registration functions that allow both teachers and students to access the program, it also holds the backToHome function which sends the user back to the main menu, the searchCredentials function which checks if the users inputted credentials exist in the program, and the pushToArray function which checks if the credentials a user has inputted already exists in the system.

menuGUI.cpp

```
START
     FUNCTION printTMenu()
          CALL printMenu("TEACHER MENU", teachMenu,
          teachMENU_ITEMS)
     END FUNCTION

     FUNCTION teachMain()
          SET isRunning TO true
          WHILE isRunning
             CALL printTMenu()
             SET selected TO selection(teachMENU_ITEMS,
             printTMenu)
             SWITCH selected
                CASE 0: CALL setTask(mainCalendar); wait(2)
                CASE 1: CALL showTasks(mainCalendar); wait(2)
                CASE 2: CALL clearTasks(mainCalendar); wait(2)
                CASE 3: CALL printCalendar(mainCalendar);
                wait(10)
```

```
                CASE 4: CLEAR SCREEN; CALL opening(); isRunning
                = false; wait(2)
            END SWITCH
        END WHILE

    FUNCTION printSMenu()
         CALL printMenu("STUDENT MENU", studentMenu,
         studentMENU_ITEMS)

    FUNCTION studentMain()
        SET isRunning TO true
        WHILE isRunning
            CALL printSMenu()
            SET selected TO selection(studentMENU_ITEMS,
            printSMenu)
            SWITCH selected
               CASE 0: CALL showTasks(mainCalendar); wait(2)
               CASE 1: CALL printCalendar(mainCalendar);
               wait(10)
            CASE 2: CLEAR SCREEN; CALL opening(); isRunning END
            SWITCH
        END WHILE
STOP
```

Figure 11: Pseudo code of the menuGUI.cpp file

This figure shows the flow of the menuGUI.cpp file, it shows how the two separate menus for the teacher and student will determine which option is selected using a switch case located inside a while loop with the condition of isRunning. Each function's switch-case is based on the amount that is assigned to an array that holds the options for each menu; for the teachers it holds 5, for the students it holds 3. Each case calls a function related to the option that was selected by the user.

main.cpp

```
START
      SET schedInfo mainCalendar [12][6][7]
      INT main
            CALL createCalendar(mainCalendar)
            CALL opening()
            RETURN 0
END
```

Figure 12: Pseudo code of the main.cpp file

This represents the main cpp file of the project, The code first sets a structure with its member values located in an array called mainCalendar, it then initializes the main function. Inside the main function it calls two functions called createCalendar, a function located in calendar.cpp with its condition set to mainCalendar, next it calls the function called opening a function located in credentials.cpp to print out the navigation menu that allows the user to select

misc.cpp

```
START
      SET currRow to 0;

      FUNCTION wait(time):
          pause program for 'time' seconds
          WHILE any key pressed:
              clear input
          END WHILE

      FUNCTION printHeader(title):
          PRINT "======================================"
          PRINT title (centered)
          PRINT "======================================"
```

```
FUNCTION isNumInRange(prompt, inputRef, minVal, maxVal):
    REPEAT
        PRINT prompt
        READ inputRef
        IF inputRef < minVal OR inputRef > maxVal:
            PRINT "Error, Number Out of Range"
        END IF
    UNTIL inputRef is within range

FUNCTION printMenu(menuName, menuItems[], numItems):
    CLEAR screen
    PRINT "===== " + menuName + " ====="
    FOR i = 0 TO numItems - 1:
        IF i == currRow:
            PRINT "● " + menuItems[i]
        ELSE:
            PRINT "  " + menuItems[i]
        END IF
    END FOR
    PRINT "Use UP/DOWN to navigate, ENTER to select"

FUNCTION selection(numItems, printFunc)
    WHILE true
        IF key is pressed:
            key = get key input
            IF key == UP_ARROW:
                currRow--
                IF currRow < 0: currRow = numItems - 1
            ELSE IF key == DOWN_ARROW:
                currRow++
                IF currRow >= numItems: currRow = 0
```

```
                ELSE IF key == ENTER:
                    selected = currRow
                    currRow = 0
                    RETURN selected
                END IF
                CALL printFunc()  // refresh menu display
            END IF
        END WHILE

    FUNCTION exitProg()
        CLEAR screen
        PRINT "Exited Program"
        TERMINATE program
END
```

Figure 13: Pseudo code of the misc.cpp file

The figure represents the workflow of the misc.cpp file, it shows all the functions that are unrelated to the functions in the other cpp files, It shows the wait function, the printHeader function which prints the format for the header of the program, the printMenu function which outputs the user-interactive menu, the selection menu which is used to determine the menu the user has selected, and the exitProg function which is used to exit the program.

Data Dictionary

The data dictionary provides a detailed description of the data types used within the code. It defines the type, size, and function of each variable and data structure utilized. The data dictionary ensures clarity in data handling and consistency throughout program development. It also serves as a reference for understanding how information is represented, stored, and manipulated within the system.

The table below presents the variables and arrays that are critical to the system. These data are equally important, although they perform different functions. Understanding their purpose is essential for learning how the code works.

Table 1: Data Dictionary

Data Name	Size	Data Type	Description
1. taskName	50 bytes	string	Holds the task's name. Is a member of the struct taskInfo
2. deadline	8 bytes	string	Holds the time of the deadline. Is a member of the struct taskInfo
3. day	4 bytes	int	Numerical representation of the day. For example, Monday = 1
4. dayName	9 bytes	string	Stores the name of the day. For example, Monday = 1
5. tasks	580 bytes	taskInfo	Stores a list of tasks (taskInfo) assigned to the day.
6. credSize	4 bytes	const int	Maximum number of credentials that can be stored.
7. openMenuSize	4 bytes	const int	Number of items in the opening menu.
8. openMenuItems	17 bytes	string	Array storing the names of the opening menu options ("Register", "Log-in", "Exit").

Data Name	Size	Data Type	Description
9. studentCreds	20000 bytes	Credentials[]	Array of student login credentials.
10. teachCreds	20000 bytes	Credentials[]	Array of teacher login credentials.
11. mainCalendar	60480 bytes	taskInfo	Array holds the data of the calendar and tasks.

Code

This section presents the header files of the code of the program, which is written in C++. The program follows a modular design approach, where the code is divided into multiple .cpp files and corresponding header files. This structure enhances the clarity, maintainability of the project by separating different components and functionalities. This modular organization not only improves readability but also simplifies debugging, testing, and future expansion of the system.

```
#include "calendar.hpp"
#include "menuGUI.hpp"
#include "credentials.hpp"

// ------- Important Arrays ------- //

Credentials studentCreds[credSize];
Credentials teachCreds[credSize];
schedInfo mainCalendar[12][6][7];

// ------- Task CRUD ------- //

int main() {
    createCalendar(mainCalendar);
    opening();
    return 0;
}
```

Figure 14. Code of “main.cpp”.

Figure 14 presents how the main function initializes the program. The code first initializes the important arrays used for the storage of tasks and storage of credentials. The main function then first formats the calendar array. After this it calls the opening() function to set up the main menu and accept user inputs.

```
#ifndef CALENDAR_HPP
#define CALENDAR_HPP

#include "misc.hpp"

// ------- Structures ------- //

struct taskInfo {
    string taskName;
    string deadline;
};

struct schedInfo {
    int day;
    string dayName;
    vector<taskInfo> tasks;
};

// ------- Function Prototypes ------- //
bool isLeapYear(int year);
int getDaysInMonth(int month);
string getDayName(int dayOfWeek);
int getStartDay(int month);
void createCalendar(schedInfo calendar[12][6][7]);  //Formats calendar
void printCalendar(schedInfo toBePrinted[12][6][7]); //Print calendar in a formatted manner
bool validDate(int m, int d);
schedInfo* searchCalendarDate(schedInfo calendar[12][6][7], string message); //returns the index on the
array based on the inputted date
void setTask(schedInfo calendar[12][6][7]);
void showTasks(schedInfo calendar[12][6][7]);
void clearTasks(schedInfo calendar[12][6][7]);

#endif
```

Figure 15: calendar.hpp

This module implements the system's task scheduling functionality through a three-dimensional array that simulates a calendar structure. Inside this module, there are two data structures defined. The first structure,taskInfo, contains two string members, taskName and deadline. The second structure , schedInfo, includes members that are an integer named day, a string named dayName, and a vector of taskInfo objects that store multiple task records associated with a specific day. The module also contains several

functions, which are setTasks(), showTasks(), andclearTasks(),that respectively handle the creation, display, and removal of task entries. These functions are mostly used in the menuGUI.cpp

```
#ifndef CREDENTIALS_HPP
#define CREDENTIALS_HPP

#include "misc.hpp"
#include "menuGUI.hpp"

const int credSize = 100;

const int openMenuSize = 3;

const string openMenuItems[openMenuSize] = {
        "Register",
        "Log-in",
        "Exit"
};

struct Credentials {
    string username;
    string password;
};

extern Credentials studentCreds[credSize];
extern Credentials teachCreds[credSize];

void searchCreds(string username, string password, int mode);
void login();
void pushToArray(string username, string password, int mode);
void registerCreds();
void printOpeningMenu();
void opening();

#endif
```

Figure 16: credentials.hpp

Figure 16 shows the code that is used to manage the storing, accessing of user data. It utilizes an array of credentials structures, which stores a username and a password as strings. The array supports up to one hundred entries. This allows multiple user accounts to be stored in it. The module also includes the functions pushToArray() which is utilized to store the credentials inputted by the user for registration and the

searchArray() for checking the array for the credentials inputted by the user for the login functionality.module also includes the bringBackHome() function, which returns the user to the main menu interface following authentication procedures. The module also includes the bringBackHome() function, which returns the user to the main menu interface following authentication procedures.

```
#ifndef MENUGUI_HPP
#define MENUGUI_HPP

#include "calendar.hpp"
#include "credentials.hpp"
#include "misc.hpp"

extern schedInfo mainCalendar[12][6][7];

const int teachMENU_ITEMS = 5;
const int studentMENU_ITEMS = 3;

const int showTaskTime = 5;

const string teachMenu[teachMENU_ITEMS] = {
    "Add Task",
    "View Tasks",
    "Delete Tasks",
    "View Calendar",
    "Logout"
};

const string studentMenu[studentMENU_ITEMS] = {
    "View Tasks",
    "View Calendar",
    "Logout"
};


void printTMenu();
void teachMain();
void printSMenu();
void studentMain();

#endif
```

Figure 17: menuGUI.hpp

Figure 17 shows the module that is responsible for managing the graphical user interface components used by both teachers and students. Both printTMenu() and printSMenu() function have the same purpose, it is called to repeatedly print the menu when a user chooses a new option, creating a cleaner interface. Moreover, teachMain() and studentMain () have pretty much the same functions, it interprets menu selections made by the user and calls the corresponding functions based on the selected option for teacher and student respectively. The menu options are stored inside a string array named teachMenu and studentMenu.

```
#ifndef MISC_HPP
#define MISC_HPP

#include <iostream> //input output
#include <iomanip> //formatting
#include <vector> //vector
#include <string> //string functions
#include <sstream> // breaking strings
#include <conio.h>  // For _kbhit() and _getch()
#include <thread> // for wait function
#include <chrono> // for wait function

using namespace std;

void wait(int time);
void printHeader(const string &title);
void isNumInRange(string dialogue, int &input , int minVal, int maxVal);
void printMenu(const string &menuName,const string menuItems[],int numItems);
int selection(int menuItems,void (*printFunc)());
void exitProg();

#endif
```

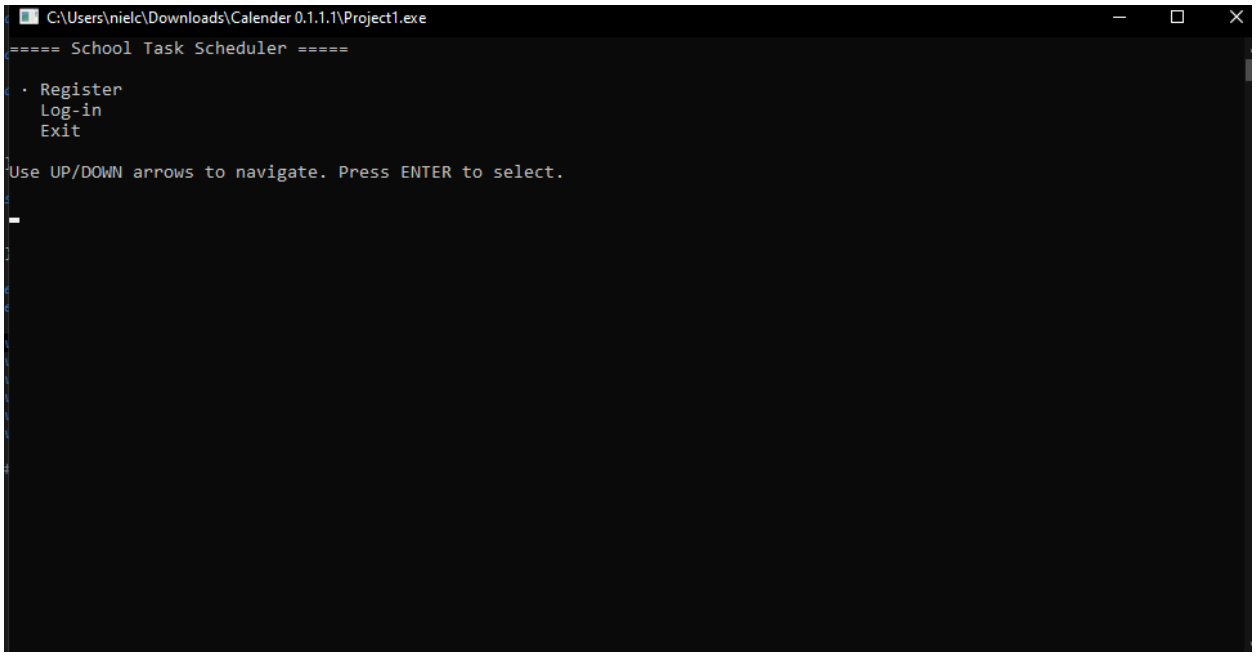
Figure 16: misc.hpp

The figure above presents the module that contains different utility functions that are used throughout the different modules in the program. Included in this modules are the following modules: iostream for input and output functions; iomanip for formatting the printed outputs; vector library for the utilization of vector functions; sstream using for breaking up strings using a separator or a delimiter; conio.h for more versatile keyboard input functions; thread and chrono libraries for the wait function. The utility functions declared in the program include: wait(), which yields the thread ,essentially stopping the program for a set time; printMenu(), which displays menu options; selection() function processes user input from arrow key navigation and selection; and exitProg() which terminates the application.

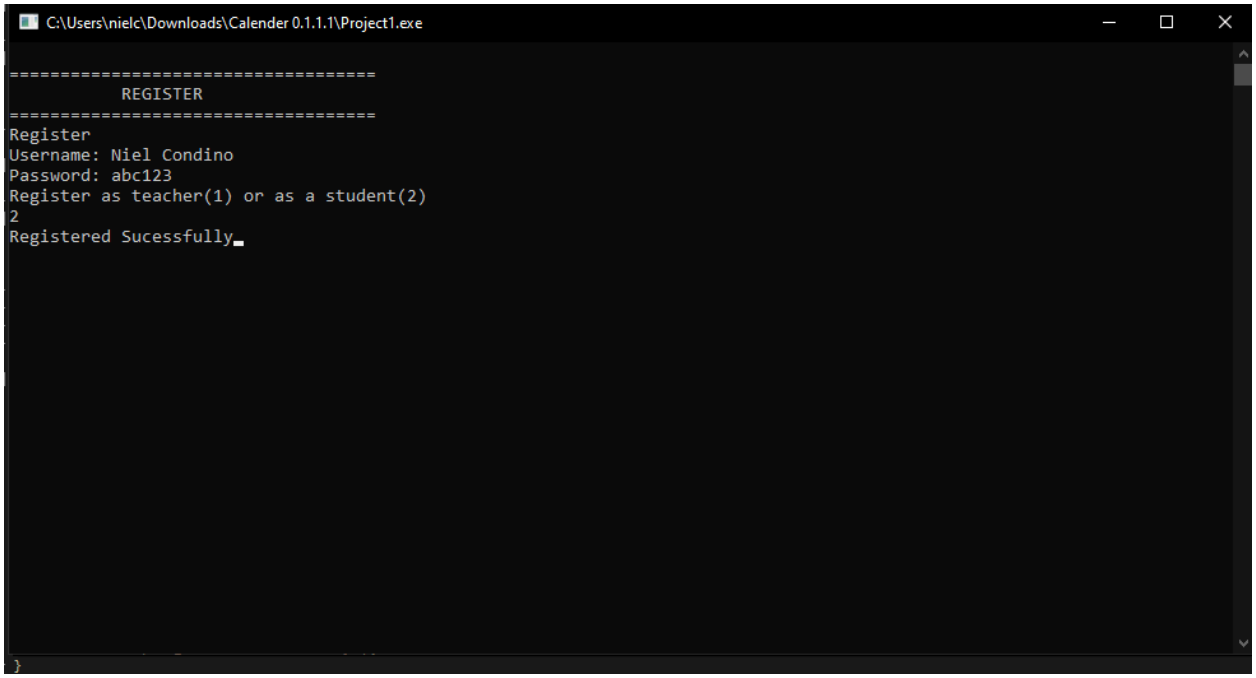
Results and Discussion

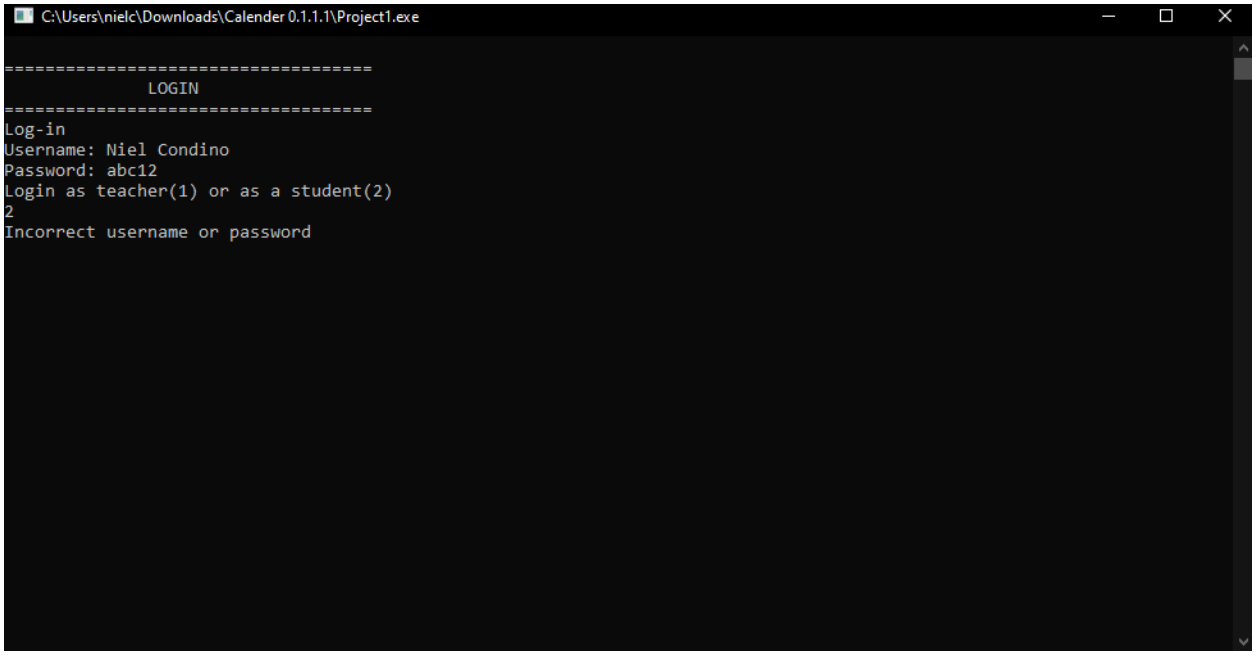
This section presents and analyzes the outcomes of the developed schedule management system. It discusses the functionality and usability of the system based on the implemented module, including user authentication, scheduling, and task viewing. The results demonstrate how each module contributes to achieving the project's objectives. Any discrepancies, limitations, or unexpected behaviors are analyzed to identify potential causes and areas for improvement.

Upon execution the program successfully displays the main menu using the printMenu() function. The menu allows the user to register, log in, or exit the program which indicates that the initialization of the program is running without any compilation errors. The menu system allows for interactive navigation using arrow keys to select an option from the menu, meaning that the selection() function works as intended.



The functions in the credentials.cpp also successfully store and access user credentials. Picking the register option allows the user to enter their own username and password which this module stores in an array of credential structures that can hold up to 100 users. During the login process, the program compares the entered credentials against the stored records. It would proceed to check the input of the user for an exact match with the credentials stored. Invalid credentials trigger an error catching system which demonstrates that the implemented error detection is working as intended. This confirms the reliability of the authentication process and keeps unauthorized users from accessing the system.





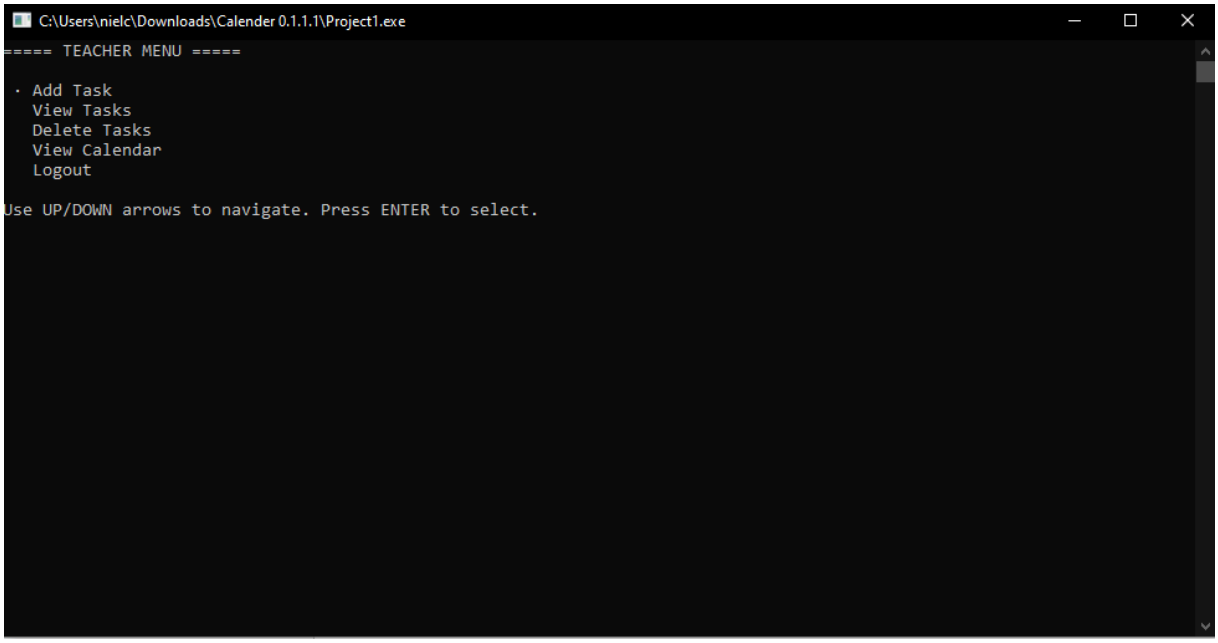
The functions inside calendar.cpp accurately generate a calendar using a 3D array of type schedInfo structured according to the specified year. The createCalendar() function formats a schedInfo array of dimensions 12×6×7 into a properly structured calendar using functions such as getDaysInMonth() to correctly check the number of days in a month, getStartDay() which uses the Zeller's congruence to get the starting day, getDayName() to correctly get the name of that day, and isLeapYear() to check if the year is a leap year. Testing by printing the array verified that all months display the correct number of days. The printed calendar format clearly distinguishes days with scheduled tasks by enclosing them in brackets , for example [15]. This design ensures that users can easily identify dates with scheduled tasks.

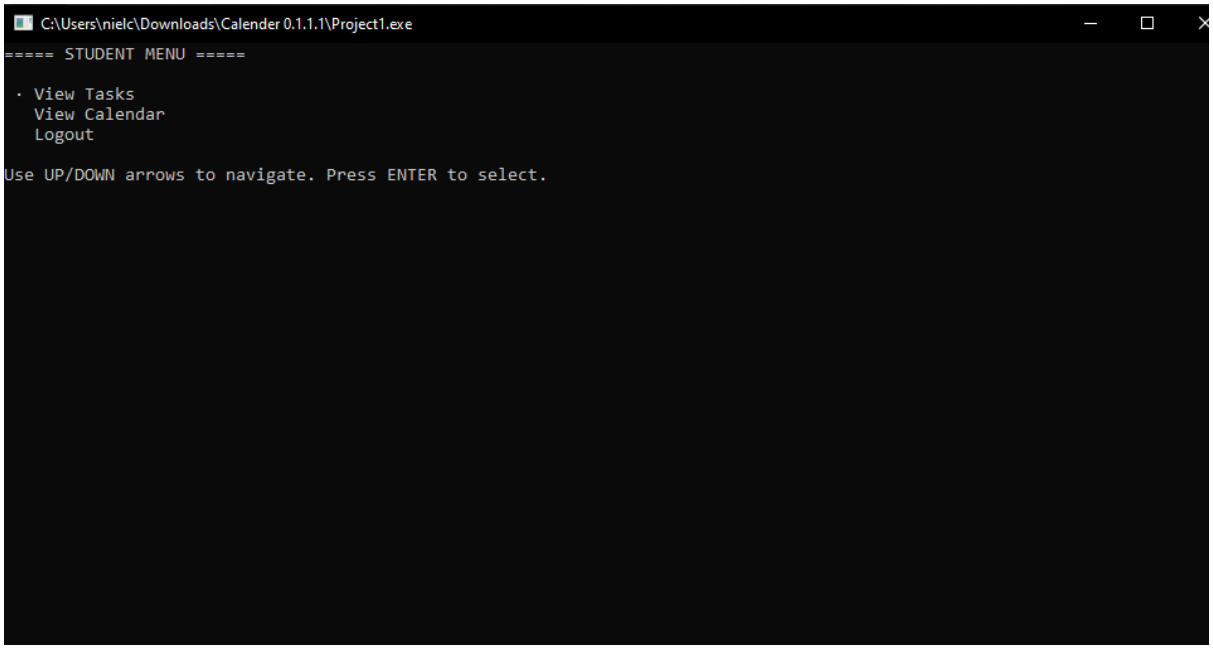
```
Month 1:
Su  Mo  Tu  We  Th  Fr  Sa
              1 [2]   3   4
   5   6   7   8   9  10  11
  12  13  14  15  16  17  18
  19  20  21  22  23  24  25
  26  27  28  29  30  31

Month 2:
Su  Mo  Tu  We  Th  Fr  Sa
                          1
   2   3   4   5   6   7   8
   9  10  11  12  13  14  15
  16  17  18  19  20  21  22
  23  24  25  26  27  28

Month 3:
Su  Mo  Tu  We  Th  Fr  Sa
                          1
   2   3   4   5   6   7   8
   9  10  11  12  13  14  15
  16  17  18  19  20  21  22
  23  24  25  26  27  28  29
  30  31

Month 4:
Su  Mo  Tu  We  Th  Fr  Sa
          1   2   3   4   5
   6   7   8   9  10  11  12
  13  14  15  16  17  18  19
  20  21  22  23  24  25  26
  27  28  29  30

Month 5:
Su  Mo  Tu  We  Th  Fr  Sa
                  1   2   3
   4   5   6   7   8   9  10
  11  12  13  14  15  16  17
  18  19  20  21  22  23  24
  25  26  27  28  29  30  31

Month 6:
Su  Mo  Tu  We  Th  Fr  Sa
   1   2   3   4   5   6   7
   8   9  10  11  12  13  14
  15  16  17  18  19  20  21
  22  23  24  25  26  27  28
  29  30

Month 7:
Su  Mo  Tu  We  Th  Fr  Sa
          1   2   3   4   5
   6   7   8   9  10  11  12
  13  14  15  16  17  18  19
  20  21  22  23  24  25  26
  27  28  29  30  31
```

```
Month 8:
Su  Mo  Tu  We  Th  Fr  Sa
                      1   2
   3   4   5   6   7   8   9
  10  11  12  13  14  15  16
  17  18  19  20  21  22  23
  24  25  26  27  28  29  30
  31

Month 9:
Su  Mo  Tu  We  Th  Fr  Sa
      1   2   3   4   5   6
   7   8   9  10  11  12  13
  14  15  16  17  18  19  20
  21  22  23  24  25  26  27
  28  29  30

Month 10:
Su  Mo  Tu  We  Th  Fr  Sa
              1   2   3   4
   5   6   7   8   9  10  11
  12  13  14  15  16  17  18
  19  20  21  22  23  24  25
  26  27  28  29  30  31

Month 11:
Su  Mo  Tu  We  Th  Fr  Sa
                          1
   2   3   4   5   6   7   8
   9  10  11  12  13  14  15
  16  17  18  19  20  21  22
  23  24  25  26  27  28  29
  30

Month 12:
Su  Mo  Tu  We  Th  Fr  Sa
      1   2   3   4   5   6
   7   8   9  10  11  12  13
  14  15  16  17  18  19  20
  21  22  23  24  25  26  27
  28  29  30  31
```

Also included in this module are the functions setTask(), showTasks(), and clearTasks(). These functions are used to set, display, and remove tasks, respectively. The performed test confirmed that these functions perform as intended. The validation function validDate() ensures that the user's entered date does not exceed the valid calendar range.

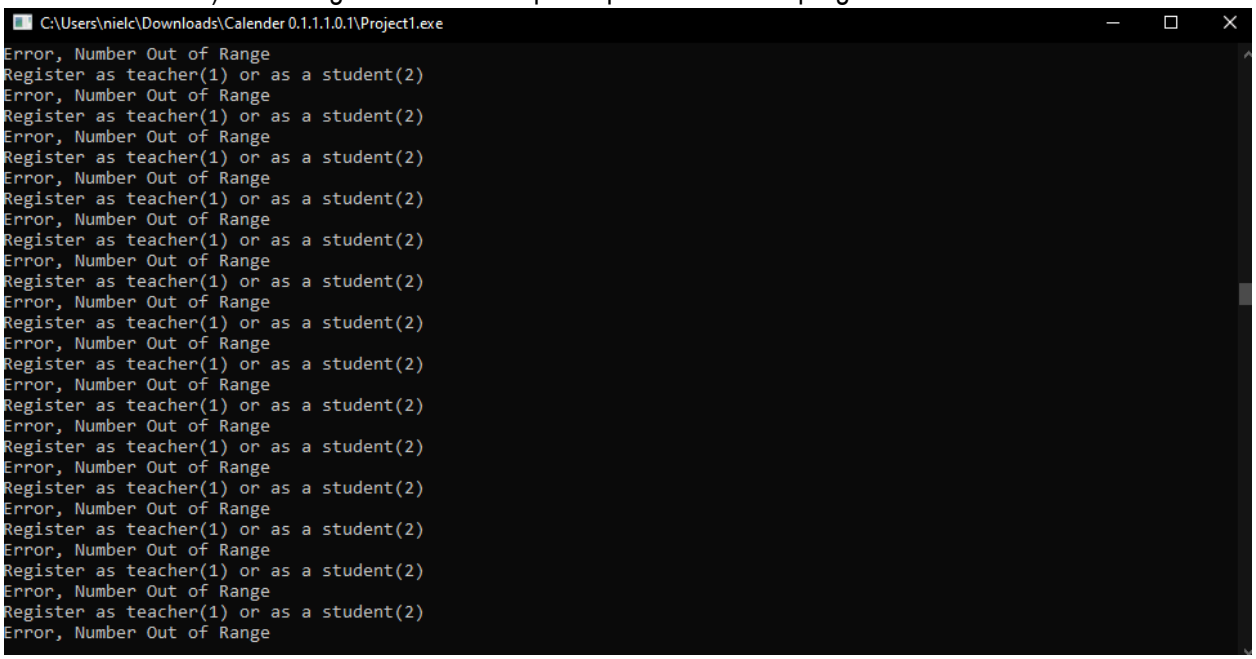
Another component is the menuGUI.cpp module, which manages the graphical user interface for both teacher and student menus. It processes user input through the selection() function in the misc.cpp module and calls the appropriate functions based on the chosen option. This design enhances user experience by providing an organized and intuitive navigation that enhances overall user experience.





The misc.cpp module is where functions like wait(), selection(), and printMenu() are used to operate the program properly. If any of these functions fail, such as selection() not returning the user's choice correctly, the program's menu navigation could become unresponsive, potentially affecting overall system performance as a result. Given their important role in maintaining smooth program operation, these functions must be thoroughly tested. Failures in these functions would compromise both usability and system performance; if the program runs as intended, it indicates that these functions are operating correctly.

However, we didn't run through any compilation errors; there still exist bugs within the program. One issue we found while testing the program is that when the user inputs a string data type in a function that only accepts an int data type, the program will repeatedly display an error message instead of prompting the user to re-enter the value (the value being 1 or 2, depending if the user is logging in as a student or teacher). Resulting in an infinite loop that prevents further progress.



An issue that occurs during our testing is that the program occasionally crashes for unknown reasons. These crashes occur at random and have not been consistently reproducible, making it difficult to debug. Additionally, the program does not retain any of the data that has been entered between sessions, as all information is stored only in memory rather than in permanent storage, which limits long-term usability.

Conclusion

The findings indicate that the program functioned as intended and successfully met its objectives. The created system allows teachers and students to register and log in, with teachers having the ability to set, view, and reschedule tasks for students. Moreover, the system allows students to view their pending assignments, assisting them in managing their schedules more efficiently. The developers also modularized the code by dividing it into multiple .cpp files, where each one is dedicated to a specific function or set of parameters to run the program properly. Doing so improves the readability and makes it easier to debug if an error ever occurs.

The developers also tested and evaluated the program's performance and accuracy. During testing, the developers identified bugs within the program. These bugs include an infinite loop triggered by entering an incorrect data type and occasional crashes while using the program with no consistent steps to reproduce them.

After further analysis the developers of this program have several recommendations for others that will create similar projects. First, implementing an input validation mechanism to prevent infinite print loops caused by incorrect data types. We also recommend creating an external storage system which lets the program remember tasks stored between different sessions when it was used. Another feature the developers recommend is creating a username and password changing feature, letting users change their credentials to have a more secured account. Incorporating a section registration option for the students where they can register to their respective section. As a final recommendation, the system should restrict task deletion privileges so that tasks can only be deleted by the teacher who created it, maintaining data integrity and accountability.

References

Dr. Gordon, R. (2023, May 16). *How to Focus on Homework to Get It Done on Time.* https://www.apu.apus.edu/area-of-study/education/resources/how-to-focus-on-homework-to-get-it-done-on-time/

LCSW. Vallejo, M. (2025, May 28). *Academic pressure: causes, effects, and coping strategies*. Mental Health Center Kids. https://mentalhealthcenterkids.com/blogs/articles/academic-pressure#:~:text=Excessive%20levels%20of%20academic%20pressure,to%20ensure%20their%20well%2Dbeing.

Zhang, C. et al. (2022, March 5). *Associations Between Academic Stress and Depressive Symptoms Mediated by Anxiety Symptoms and Hopelessness Among Chinese College Students.* Associations Between Academic Stress and Depressive Symptoms Mediated by Anxiety Symptoms and Hopelessness Among Chinese College Students - PMC

Appendices

Appendix A: calendar.cpp

```
#include "calendar.hpp"

/*
================================
Calendar creation essentials
================================
*/

const int year = 2025;

// Check for leap year
bool isLeapYear(int year) {
    return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
}

// Get number of days in a given month
int getDaysInMonth(int month) {
    const int daysInMonth[12] = {31,28,31,30,31,30,31,31,30,31,30,31};
    // February
    if (month == 1 && isLeapYear(year)){
        return 29;
    }
    return daysInMonth[month];
}

string getDayName(int dayOfWeek){
    switch (dayOfWeek){
        case 0:
            return "Sunday";
        case 1:
            return "Monday";
        case 2:
            return "Tuesday";
        case 3:
            return "Wednesday";
        case 4:
            return "Thursday";
```

```
        case 5:
            return "Friday";
        case 6:
            return "Saturday";

    }
    return " ";
}

// Compute which weekday the month starts on (0 = Sunday, 6 = Saturday)
int getStartDay(int month) {
    // Zeller's congruence from online
    int m = month + 1;
    int y = year;
    if (m < 3) { m += 12; y--; }
    int q = 1;
    int k = y % 100;
    int j = y / 100;
    int h = (q + (13*(m + 1))/5 + k + (k/4) + (j/4) + (5*j)) % 7;
    return (h + 6) % 7; // 0 is sunday ,6 is saturday
}

void createCalendar(schedInfo calendar[12][6][7]){
    // 12 months, 6 weeks ,7 days

    for (int month = 0; month < 12; month++) {
        int startDay = getStartDay(month);
        int days = getDaysInMonth(month);

        int week = 0;
        int dayOfWeek = startDay;

        for (int day = 1; day <= days; day++) {
            schedInfo *currentDay = &calendar[month][week][dayOfWeek];
            currentDay -> day = day;
            currentDay -> dayName = getDayName(dayOfWeek);

            dayOfWeek++;

            if (dayOfWeek > 6) {
```

```
            dayOfWeek = 0;
            week++;// move to next week
        }
    }
  }
}

/*
============================================================
Calendar utilization
============================================================
*/

void printCalendar(schedInfo toBePrinted[12][6][7]) {
   for (int month = 0; month < 12; ++month) {
      cout << "Month " << month + 1 << ":\n";
      cout << "Su  Mo  Tu  We  Th  Fr  Sa\n";

      for (int week = 0; week < 6; week++) {
         bool emptyWeek = true;
         for (int day = 0; day < 7; day++) {
            if (toBePrinted[month][week][day].day != 0){
               emptyWeek = false;
            }
         }

         if (emptyWeek) break; // skip empty weeks at end

         for (int day = 0; day < 7; day++) {
            if (toBePrinted[month][week][day].day == 0){
               cout << setw(4) << " ";  // keep spacing consistent
            }
            else{
               if (toBePrinted[month][week][day].tasks.empty()){
                  cout << setw(4) << toBePrinted[month][week][day].day;//bold
               }
               else {
                  cout << setw(4) << "[" + to_string(toBePrinted[month][week][day].day) + "]" ;
               }
```

```
            }
        }
        cout << "\n";
    }
    cout << "\n";
  }
}

bool validDate(int m,int d){
  if ((m > 12 || m <= 0) || (d > getDaysInMonth(m-1) || d <= 0)){
    return false;
  }
  return true;
}

schedInfo* searchCalendarDate(schedInfo calendar[12][6][7],string message){
  int m,d;
  bool isDateValid;
  do {
    cout << message;
    cout << "Month:";
    cin >> m;
    cout << "Day: ";
    cin >> d;
    isDateValid = validDate(m,d);
    if (!isDateValid){
      cout << "Invalid date\n";
    }
    cin.ignore();//Remove endline in the buffer
  }while(!isDateValid);

  for (int i = 0; i < 6; i++){//week
    for (int j = 0; j < 7;j++){//day
      if (calendar[m-1][i][j].day == d){
        return &calendar[m-1][i][j];
      }
    }
  }
  return nullptr;
}
```

```
CRUD utility
=============================================================
*/

/*
============================================================

void setTask(schedInfo calendar[12][6][7]){
    int countTasks;

    schedInfo* chosenDate = searchCalendarDate(calendar,"Select month and day for deadline:\n");
    isNumInRange("Enter task amount: ", countTasks , 1, 10);

    for (int i = 1; i <= countTasks;i++){
        cout << endl;
        string taskName;
        cout << "Enter task name: ";
        getline(cin,taskName);
        string deadlineTime;
        cout << "Enter deadline time: ";
        getline(cin,deadlineTime);
        chosenDate -> tasks.push_back({taskName,deadlineTime});
        cout << endl;
    }
    cout << "\n=================== Task Successfully Added ====================\n";
}

void showTasks(schedInfo calendar[12][6][7]){
    schedInfo* chosenDate = searchCalendarDate(calendar,"Show tasks from the date...\n");
    cout << "Available Task " <<" (" << chosenDate->dayName << "):\n";
    if (chosenDate->tasks.empty()) {
        cout << "  No tasks scheduled.\n";
    } else {
        for (const auto &task : chosenDate->tasks) { //Loops through the vector of the tasks
            cout << "  - " << task.taskName << " (Deadline time: " << task.deadline << ")\n";
        }
    }
}

void clearTasks(schedInfo calendar[12][6][7]) {
```

```
    schedInfo* chosenDate = searchCalendarDate(calendar, "Select task to be deleted\n");
    if (chosenDate->tasks.empty()) {
        cout << "  No tasks scheduled.\n";
        return;
    }
    for (size_t i = 0; i < chosenDate->tasks.size(); i++)
        cout << "(" << i + 1 << ") " << chosenDate->tasks[i].taskName << " (Deadline: " <<
chosenDate->tasks[i].deadline << ")\n";

    cout << "\n To be deleted(if multiple, separate each with comma(,) and no spaces.Example :1,2) : ";
    string tbDeleted;
    getline(cin, tbDeleted);//get raw string
    stringstream ss(tbDeleted);
    string temp;
    vector<int> toDelete;//storage of the collected index
    while (getline(ss, temp, ',')) { //filters the string, using ',' as a separator
        int idx = stoi(temp);//turn string to int
        if (idx >= 1 && idx <= (int)chosenDate->tasks.size()){
            toDelete.push_back(idx - 1);//collect the index
        }
    }

    // delete in reverse order
    for (int i = toDelete.size() - 1; i >= 0; i--){
        chosenDate->tasks.erase(chosenDate->tasks.begin() + toDelete[i]);
    }
    cout << "Deleted selected tasks.\n";
}
```

Appendix B:credentials.cpp

```
#include "credentials.hpp"

// ------- Bring back to homescreen function -------//
void backToHome(string message){
    cout << message;
    wait(3);
    opening();
}

// ------- Credential Search Function ------- //

void searchCreds(string username, string password, int mode) {
    if (username.empty() || password.empty()) { //Check if username is empty
        backToHome("Credentials cannot be empty!");
    }
    bool found = false;
    for (int i = 0; i < credSize; i++) {// Find credential in array
        if (mode == 2) {
            if (studentCreds[i].username == username) {
                found = true;
                if (studentCreds[i].password == password) {
                    studentMain();
                } else {
                    backToHome("Incorrect username or password");
                    wait(3);
                    opening();
                }
                break;
            }
        } else {
            if (teachCreds[i].username == username) {
                found = true;
                if (teachCreds[i].password == password) {
                    teachMain();
                } else {
                    backToHome("Incorrect username or password");
                }
                break;
            }
```

```
        }
    }
    if (!found) {
        backToHome("Account does not exist");
    }
}

// ------- Login Function ------- //

void login() {
    system("CLS");
    printHeader("LOGIN");
    int role;
    string username;
    string password;

    cout << "Log-in\n";
    cout << "Username: ";
    getline(cin, username);
    cout << "Password: ";
    getline(cin, password);

    isNumInRange("Login as teacher(1) or as a student(2)\n", role, 1, 2);

    searchCreds(username, password, role);
}

// ------- Registration Functions ------- //

void pushToArray(string username, string password, int role) {
    if (username.empty() || password.empty()) {
    backToHome("Credentials cannot be empty!");
    }
    for (int i = 0; i < credSize; i++) {//Find empty index
        if (role == 2) {
            if (studentCreds[i].username == username) {
                backToHome("username already exists");
            }
            if (studentCreds[i].username.empty()) {
                studentCreds[i].username = username;
```

```
            studentCreds[i].password = password;
            return;
        }
    } else {
        if (teachCreds[i].username == username) {
            backToHome("username already exists");
        }
        if (teachCreds[i].username.empty()) {
            teachCreds[i].username = username;
            teachCreds[i].password = password;
            return;
        }
    }
}

void registerCreds() {
    system("CLS");
    printHeader("REGISTER");
    int role = 0;
    string username, password;

    cout << "Register\n";
    cout << "Username: ";
    getline(cin, username);
    cout << "Password: ";
    getline(cin, password);

    isNumInRange("Register as teacher(1) or as a student(2)\n", role, 1, 2);

    pushToArray(username, password, role);
    backToHome("Registered Sucessfully");
}

// ------ Main selection screen ------ //

void printOpeningMenu() {
    printMenu("School Task Scheduler", openMenuItems, openMenuSize);
}
```

```
void opening() {
    int selected = 0;
    printOpeningMenu();
    selected = selection(openMenuSize, printOpeningMenu);
    switch (selected) {
        case 0: registerCreds(); break;
        case 1: login(); break;
        case 2: exitProg(); break;
    }
}
```

Appendix C:menuGUI.cpp

```
#include "menuGUI.hpp"

// ------- Headers ------- //
void printTMenu() {
        printMenu("TEACHER MENU",teachMenu,teachMENU_ITEMS);
}

void printSMenu() {
        printMenu("STUDENT MENU",studentMenu,studentMENU_ITEMS);
}

// ------- Selection Menu ------- //

void teachMain(){
        int selected = 0;
        bool isRunning = true;
        while (isRunning){
                printTMenu();
                int selected = selection(teachMENU_ITEMS,printTMenu);
                switch (selected){
                        case 4:
                                system("CLS");
                                isRunning = false;
                                opening();
                                break;
                    case 0:
                                setTask(mainCalendar);
```

```
                                wait(2);
                                break;
        case 1:
                                showTasks(mainCalendar);
                                wait(5);
                                break;
        case 2:
                                clearTasks(mainCalendar);
                                wait(2);
                                break;
        case 3:
                                printCalendar(mainCalendar);
                                wait(10);
                                system("CLS");
                                break;
                }
        }
}

void studentMain(){
        int selected = 0;
        bool isRunning = true;
        while (isRunning){
                printSMenu();
                int selected = selection(studentMENU_ITEMS,printSMenu);
                switch (selected){
                        case 2:
                                system("CLS");
                                isRunning = false;
                                opening();
                                break;
        case 0:
                                showTasks(mainCalendar);
                                wait(5);
                                break;
        case 1:
                                printCalendar(mainCalendar);
                                wait(10);
                                break;
                }
```

```
        }
}
```

Appendix D: misc.cpp

```
#include "misc.hpp"

int currRow = 0;

// ------- Wait Function ------- //

void wait(int time) {
	this_thread::sleep_for(chrono::seconds(time));
}

// ------- Header Format ------- //

void printHeader(const string &title) {
   cout << "\n=====================================\n";
   cout << setw(19) << title << "\n";
   cout << "=====================================\n";
}

// ------- Error Function ------- //

void isNumInRange(string dialogue, int &input , int minVal, int maxVal){
	bool invalid = true;
	do {
		cout << dialogue;
		cin >> input;
		invalid = (input  > maxVal || input < minVal);
		if (invalid){
			cout << "Error, Number Out of Range\n";
		}
	}while (invalid);
	cin.ignore();
}

// ------- Navigation Function ------- //

void printMenu(const string &menuName,const string menuItems[],int numItems){
	system("CLS");
   cout << "===== "<< menuName << " =====\n\n";
   for (int i = 0; i < numItems; i++) {
```

```
        if (i == currRow){
            cout << " " << char(249) << " " << menuItems[i] << endl;
                        }
        else{
            cout << "   " << menuItems[i] << endl;
                        }
    }
    cout << "\nUse UP/DOWN arrows to navigate. Press ENTER to select.\n\n";
}

int selection(int menuItems,void (*printFunc)()){
            int returnedRow;
            while (true) {
        if (_kbhit()) {
            int ch = _getch();
            if (ch == 0 || ch == 224) {
                ch = _getch();
                switch (ch) {
                    case 72: currRow--; if (currRow < 0) currRow = menuItems - 1; break;
                    case 80: currRow++; if (currRow >= menuItems) currRow = 0; break;
                }
                printFunc();
            } else if (ch == 13) {
             returnedRow = currRow;
             currRow = 0; //Reset back to zero
             return returnedRow;
            }
        }
    }
}

void exitProg(){
            system("CLS");
            cout << "Exited Program";
            exit(0);
}
```