Project Report

Course title: Data Structure Lab

Course code: CSE 134

<u>Submitted to:</u>

Lecturer: Rishad Amin Pulok

Metropolitan University, Bangladesh

<u>Submitted by:</u>

Team: Recursive Runners

<u>Members-</u>

1. Mehedi Hasan(232-115-005)

2.Mehedi Hasan Emon (232-115-018)

3.Mehrab Hasan Akhanjee(232-115-006)

Batch: 59th

Section: A

Department of Computer Science & Engineering Metropolitan University, Bangladesh

Date of Submission: 12-11-2024

Report on Weekly Schedule Manager Program

1. Introduction

The Weekly Schedule Manager is a C-based application designed to help users manage their weekly schedules. It allows students to add, edit, delete, and view subjects and their respective times, offering a simple yet effective way to organize their academic timetable. The program provides basic features for users to input and manage their schedules, but it is limited in some areas and has room for improvement.

2. Key Features

Add Subject: Allows users to add subjects and their corresponding times to specific days of the week.

Edit Subject: Users can modify the subject names or times for any existing entry.

Delete Subject: Provides the ability to remove subjects from a specific day and reorganize the remaining entries.

Display Day's Schedule: Users can view the subjects and times for any given day.

Display Weekly Schedule: Displays the full schedule for the entire week.

Menu-driven Interface: Users interact with the program through a simple menu, making it easy to navigate between different functionalities.

3. Working step's(for imaginary inputs):

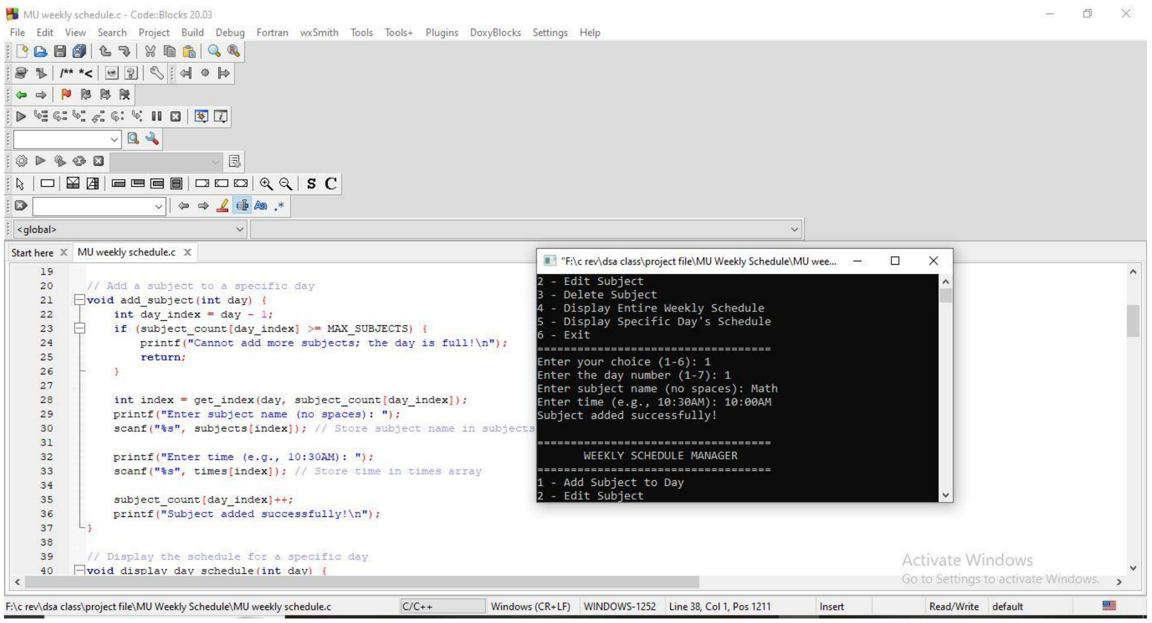
# Adding Subjects to Day 1 in the Weekly Schedule:

Let's assume the user wants to add two subjects to Day 1 (Sunday). Below is how the functions work step-by-step when adding subjects.

First Subject Addition:

- **Step 1:** The user selects option 1 from the menu to "Add Subject to Day".
- **Step 2:** The program asks the user to enter the day number.
  - The user enters `1` (day 1= Sunday)
- **Step 3:** The program checks if there is space for adding a subject to Day 1.
  - Since `subject_count[0]` is `0` initially, the program proceeds to add the subject.
- **Step 4:** The program calls the `add_subject` function with `day = 1` (Sunday).
  - Inside the `add_subject` function:
    - `day_index = 1 - 1 = 0` (0 represents Sunday).
    - The program checks if `subject_count[0]` (the count of subjects for Sunday) is less than `MAX_SUBJECTS` (10). Since it's `0`, it's fine to add a subject.

        - The program then calculates the index where the subject will be stored using `get_index(1, subject_count[0])` → `get_index(1, 0) = 0`.
        - The program asks the user to input the subject name and time. Let's assume the user enters:
            - Subject name: `Math`
            - Time: `10:00AM`
        - These values are stored in the arrays:
            - `subjects[0] = "Math"`
            - `times[0] = "10:00AM"`
        - `subject_count[0]` (subject count for Sunday) is incremented by 1.
        - The program prints: "`Subject added successfully!`"



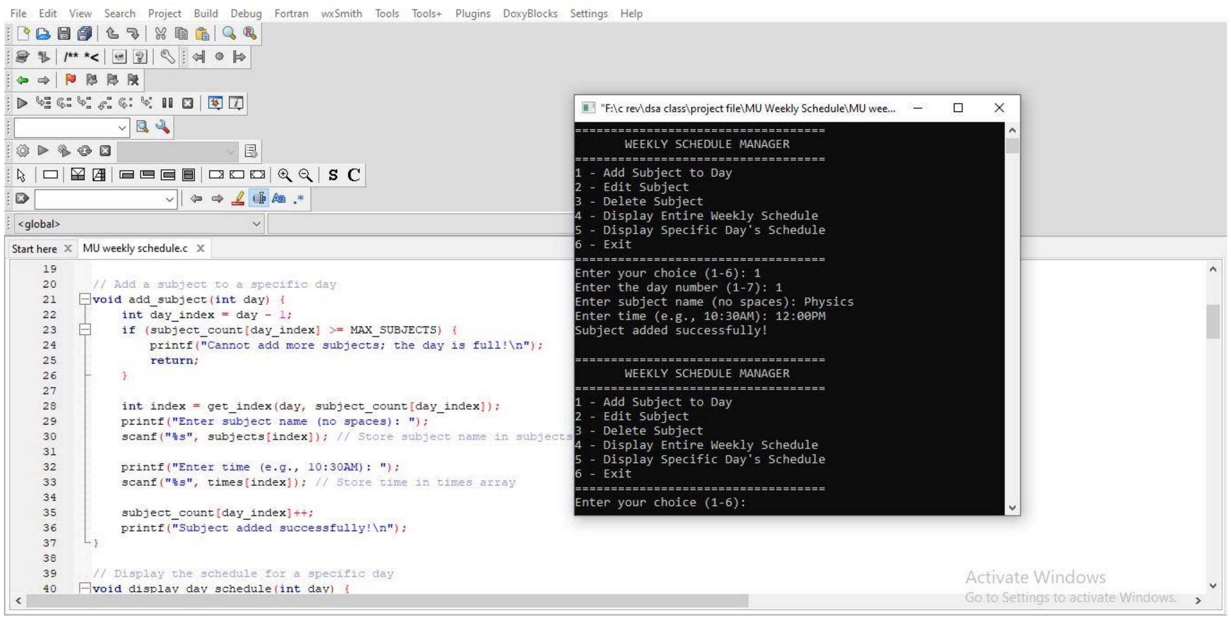
```c
// Add a subject to a specific day
void add_subject(int day) {
    int day_index = day - 1;
    if (subject_count[day_index] >= MAX_SUBJECTS) {
        printf("Cannot add more subjects; the day is full!\n");
        return;
    }

    int index = get_index(day, subject_count[day_index]);
    printf("Enter subject name (no spaces): ");
    scanf("%s", subjects[index]); // Store subject name in subjects

    printf("Enter time (e.g., 10:30AM): ");
    scanf("%s", times[index]); // Store time in times array

    subject_count[day_index]++;
    printf("Subject added successfully!\n");
}

// Display the schedule for a specific day
void display_day_schedule(int day) {
```

```
2 - Edit Subject
3 - Delete Subject
4 - Display Entire Weekly Schedule
5 - Display Specific Day's Schedule
6 - Exit
==================================
Enter your choice (1-6): 1
Enter the day number (1-7): 1
Enter subject name (no spaces): Math
Enter time (e.g., 10:30AM): 10:00AM
Subject added successfully!

==================================
       WEEKLY SCHEDULE MANAGER
==================================
1 - Add Subject to Day
2 - Edit Subject
```

Second Subject Addition:

- **Step 1:** The user selects option 1 again to add another subject.
- **Step 2:** The program asks the user to enter the day number again.
    - The user enters `1` again for Sunday.
- **Step 3:** The program checks if there's space to add another subject to Sunday.
    - Since `subject_count[0]` is now `1` (one subject already added), the program proceeds.
- **Step 4:** The program calls the `add_subject` function again with `day = 1` (Sunday).
    - Inside the `add_subject` function:
        - `day_index = 1 - 1 = 0`.
        - It checks if `subject_count[0]` is less than `MAX_SUBJECTS`. Now `subject_count[0] = 1`, so there's space for more subjects.
        - The program then calculates the index for storing the second subject: `get_index(1, 1) = 1`.

    - The program asks the user for the second subject's name and time. Let's assume the user enters:
        - Subject name: `Physics`
        - Time: `12:00PM`
    - These values are stored:
        - `subjects[1] = "Physics"`
        - `times[1] = "12:00PM"`
    - `subject_count[0]` is incremented again.
    - The program prints: `"Subject added successfully!"`



```c
// Add a subject to a specific day
void add_subject(int day) {
    int day_index = day - 1;
    if (subject_count[day_index] >= MAX_SUBJECTS) {
        printf("Cannot add more subjects; the day is full!\n");
        return;
    }

    int index = get_index(day, subject_count[day_index]);
    printf("Enter subject name (no spaces): ");
    scanf("%s", subjects[index]); // Store subject name in subjects

    printf("Enter time (e.g., 10:30AM): ");
    scanf("%s", times[index]); // Store time in times array

    subject_count[day_index]++;
    printf("Subject added successfully!\n");
}

// Display the schedule for a specific day
void display_day_schedule(int day) {
```

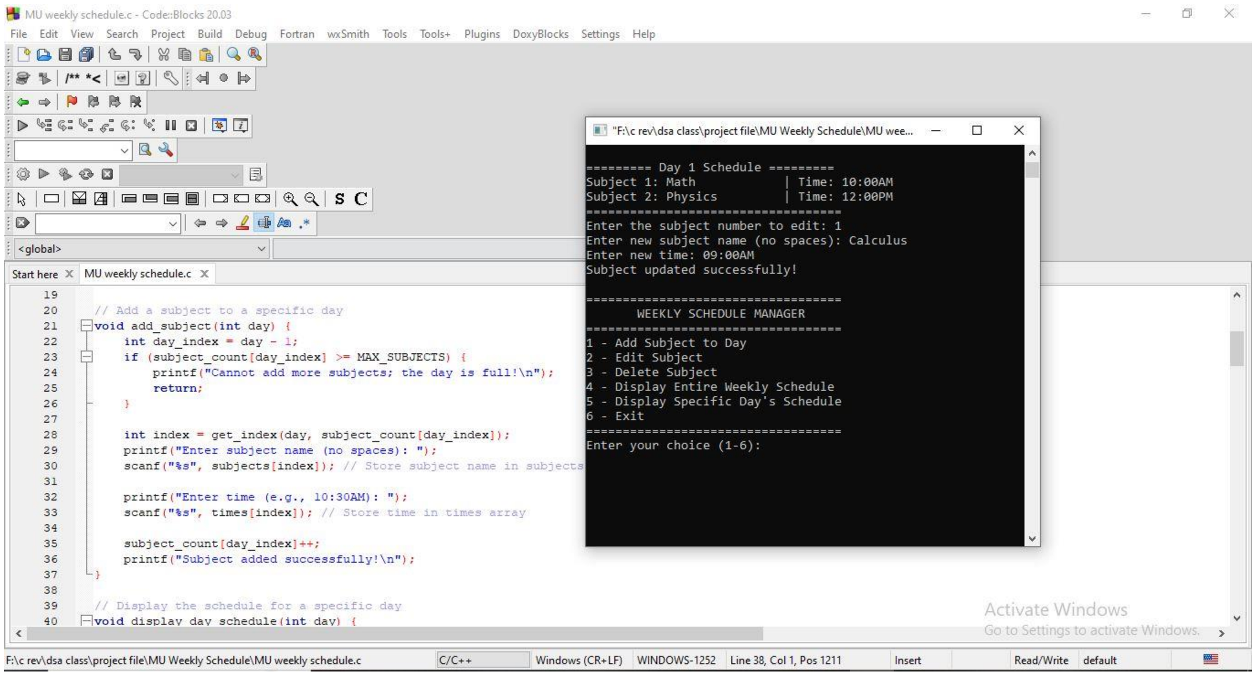
"F:\c rev\dsa class\project file\MU Weekly Schedule\MU wee...

```
===================================
       WEEKLY SCHEDULE MANAGER
===================================
1 - Add Subject to Day
2 - Edit Subject
3 - Delete Subject
4 - Display Entire Weekly Schedule
5 - Display Specific Day's Schedule
6 - Exit
===================================
Enter your choice (1-6): 1
Enter the day number (1-7): 1
Enter subject name (no spaces): Physics
Enter time (e.g., 10:30AM): 12:00PM
Subject added successfully!

===================================
       WEEKLY SCHEDULE MANAGER
===================================
1 - Add Subject to Day
2 - Edit Subject
3 - Delete Subject
4 - Display Entire Weekly Schedule
5 - Display Specific Day's Schedule
6 - Exit
===================================
Enter your choice (1-6):
```

# Editing a Subject on Day 1 (Sunday)

- **Step 1:** The user selects option 2 from the menu to "Edit Subject".
- **Step 2:** The program asks the user to enter the day number.
    - The user enters `1` for Sunday.
- **Step 3:** The program displays the current subjects for Sunday:
    - Subject 1: Math | Time: 10:00AM
    - Subject 2: Physics | Time: 12:00PM
- **Step 4:** The program asks the user to enter the subject number to edit.
    - The user enters `1` to edit "Math".
- **Step 5:** The program verifies the subject number is valid and then proceeds to edit the subject.
    - The program asks the user to input a new subject name and time. Let's assume the user enters:
        - New Subject name: `Calculus`
        - New Time: `09:00AM`
    - The subject is updated in the arrays:
        - `subjects[0] = "Calculus"`
        - `times[0] = "09:00AM"`

   - The program prints: `"Subject updated successfully!"`

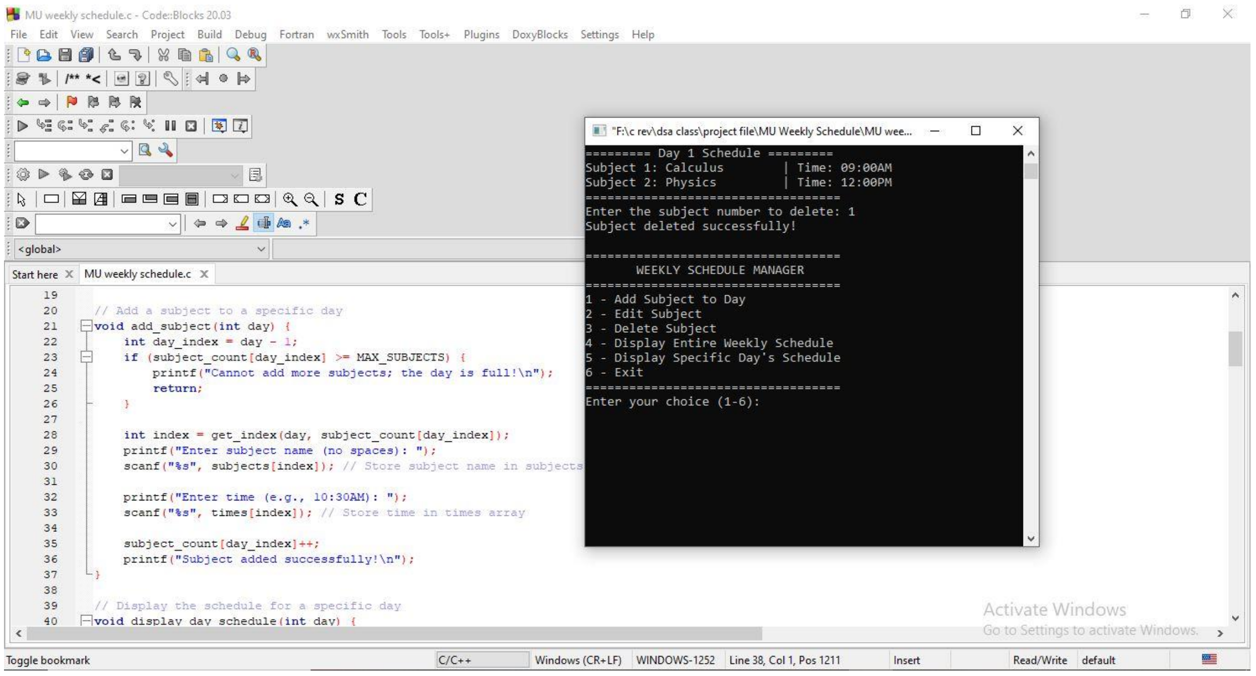


```c
// Add a subject to a specific day
void add_subject(int day) {
    int day_index = day - 1;
    if (subject_count[day_index] >= MAX_SUBJECTS) {
        printf("Cannot add more subjects; the day is full!\n");
        return;
    }

    int index = get_index(day, subject_count[day_index]);
    printf("Enter subject name (no spaces): ");
    scanf("%s", subjects[index]); // Store subject name in subjects

    printf("Enter time (e.g., 10:30AM): ");
    scanf("%s", times[index]); // Store time in times array

    subject_count[day_index]++;
    printf("Subject added successfully!\n");
}

// Display the schedule for a specific day
void display_day_schedule(int day) {
```

```
========= Day 1 Schedule =========
Subject 1: Math                | Time: 10:00AM
Subject 2: Physics             | Time: 12:00PM
===================================
Enter the subject number to edit: 1
Enter new subject name (no spaces): Calculus
Enter new time: 09:00AM
Subject updated successfully!

===================================
       WEEKLY SCHEDULE MANAGER
===================================
1 - Add Subject to Day
2 - Edit Subject
3 - Delete Subject
4 - Display Entire Weekly Schedule
5 - Display Specific Day's Schedule
6 - Exit
===================================
Enter your choice (1-6):
```

# Deleting a Subject from Day 1 (Sunday):

- **Step 1:** The user selects option 3 from the menu to "Delete Subject".
- **Step 2:** The program asks the user to enter the day number.
   - The user enters `1` (Sunday).
- **Step 3:** The program displays the current schedule for Sunday:
   - Subject 1: Math | Time: 10:00AM
   - Subject 2: Physics | Time: 12:00PM
- **Step 4:** The program asks the user to enter the subject number to delete.
   - The user enters `1` to delete "Math".
- **Step 5:** The program verifies the subject number is valid and then proceeds to delete the subject.
   - It shifts the remaining subjects up by one slot:
      - "Physics" moves to the first position, so `subjects[0] = "Physics"` and `times[0] = "12:00PM"`.
      - `subject_count[0]` is decremented.
   - The program prints: `"Subject deleted successfully!"`

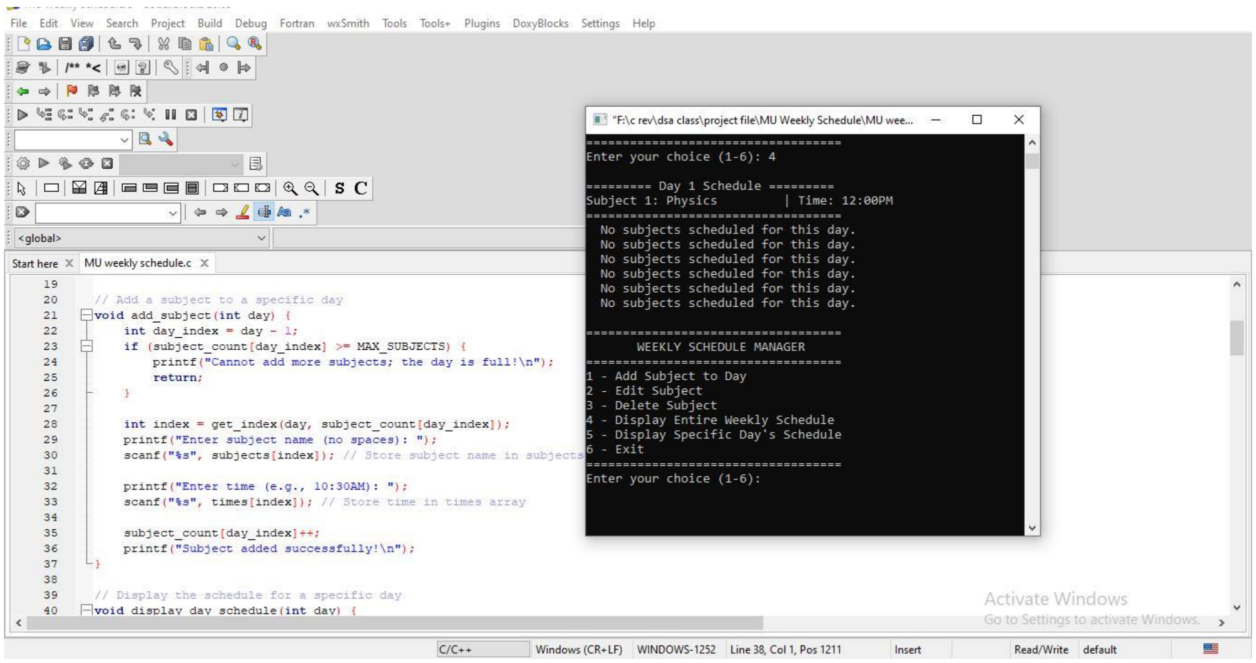


```c
// Add a subject to a specific day
void add_subject(int day) {
    int day_index = day - 1;
    if (subject_count[day_index] >= MAX_SUBJECTS) {
        printf("Cannot add more subjects; the day is full!\n");
        return;
    }

    int index = get_index(day, subject_count[day_index]);
    printf("Enter subject name (no spaces): ");
    scanf("%s", subjects[index]); // Store subject name in subjects

    printf("Enter time (e.g., 10:30AM): ");
    scanf("%s", times[index]); // Store time in times array

    subject_count[day_index]++;
    printf("Subject added successfully!\n");
}

// Display the schedule for a specific day
void display_day_schedule(int day) {
```

```
========= Day 1 Schedule =========
Subject 1: Calculus         | Time: 09:00AM
Subject 2: Physics          | Time: 12:00PM
===================================
Enter the subject number to delete: 1
Subject deleted successfully!

===================================
       WEEKLY SCHEDULE MANAGER
===================================
1 - Add Subject to Day
2 - Edit Subject
3 - Delete Subject
4 - Display Entire Weekly Schedule
5 - Display Specific Day's Schedule
6 - Exit
===================================
Enter your choice (1-6):
```

# Display Function (Display Entire Weekly Schedule)

- **Step 1:** The user selects option 4 from the menu to "Display Entire Weekly Schedule".
- **Step 2:** The program calls the `display_weekly_schedule` function.
  - Inside `display_weekly_schedule`, the program iterates through all 7 days.
    - For each day, it calls the `display_day_schedule` function to show the subjects for that day.
    - If the day has no subjects, it will print: "`No subjects scheduled for this day.`"
  - The program displays the entire schedule for the week



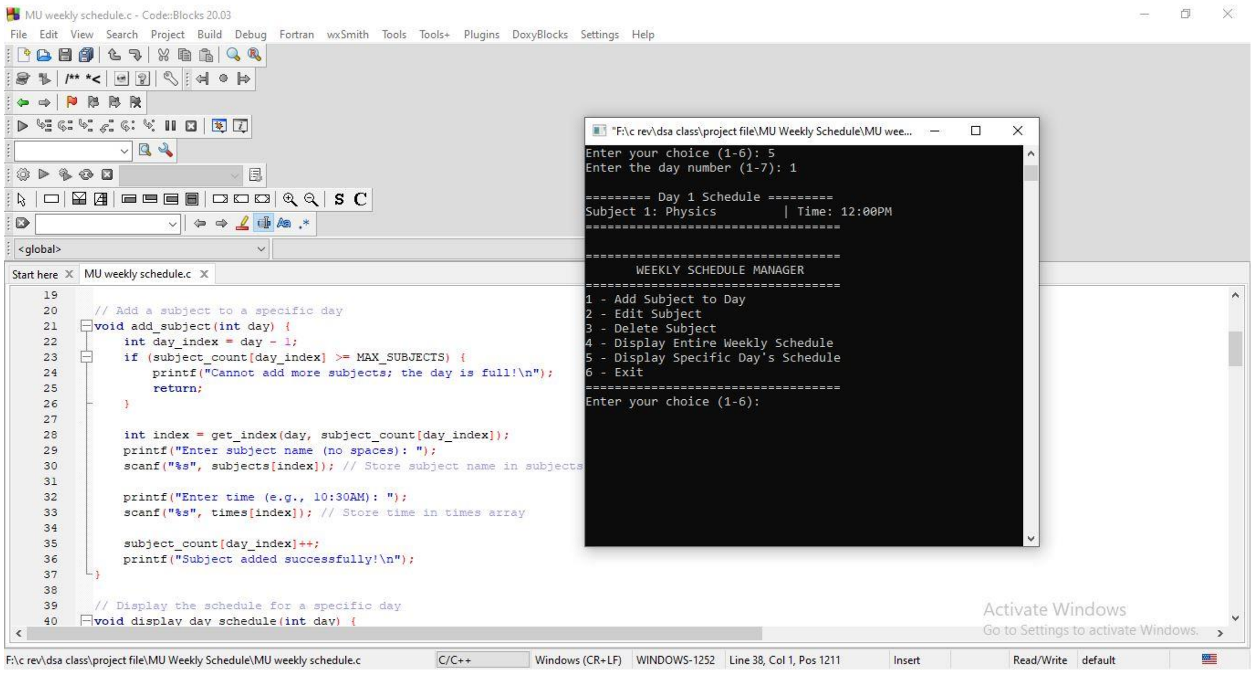
```c
// Add a subject to a specific day
void add_subject(int day) {
    int day_index = day - 1;
    if (subject_count[day_index] >= MAX_SUBJECTS) {
        printf("Cannot add more subjects; the day is full!\n");
        return;
    }

    int index = get_index(day, subject_count[day_index]);
    printf("Enter subject name (no spaces): ");
    scanf("%s", subjects[index]); // Store subject name in subjects

    printf("Enter time (e.g., 10:30AM): ");
    scanf("%s", times[index]); // Store time in times array

    subject_count[day_index]++;
    printf("Subject added successfully!\n");
}

// Display the schedule for a specific day
void display_day_schedule(int day) {
```

"F:\c rev\dsa class\project file\MU Weekly Schedule\MU wee...

```
===================================
Enter your choice (1-6): 4

========= Day 1 Schedule =========
Subject 1: Physics          | Time: 12:00PM
===================================
  No subjects scheduled for this day.
  No subjects scheduled for this day.
  No subjects scheduled for this day.
  No subjects scheduled for this day.
  No subjects scheduled for this day.
  No subjects scheduled for this day.

===================================
       WEEKLY SCHEDULE MANAGER
===================================
1 - Add Subject to Day
2 - Edit Subject
3 - Delete Subject
4 - Display Entire Weekly Schedule
5 - Display Specific Day's Schedule
6 - Exit
===================================
Enter your choice (1-6):
```

# Specific Display Function (Display a Specific Day's Schedule)

- **Step 1:** The user selects option 5 from the menu to "Display Specific Day's Schedule".
- **Step 2:** The program asks the user to enter the day number.
    - The user enters `1` for Sunday.
- **Step 3:** The program calls the `display_day_schedule` function with `day = 1`.
    - Inside `display_day_schedule`, the program checks if there are subjects for Sunday.
        - If there are subjects, it prints them.
        - If there are no subjects, it prints: "`No subjects scheduled for this day.`"
    - The program displays the schedule for that specific day (Sunday).



```c

// Add a subject to a specific day
void add_subject(int day) {
    int day_index = day - 1;
    if (subject_count[day_index] >= MAX_SUBJECTS) {
        printf("Cannot add more subjects; the day is full!\n");
        return;
    }

    int index = get_index(day, subject_count[day_index]);
    printf("Enter subject name (no spaces): ");
    scanf("%s", subjects[index]); // Store subject name in subjects

    printf("Enter time (e.g., 10:30AM): ");
    scanf("%s", times[index]); // Store time in times array

    subject_count[day_index]++;
    printf("Subject added successfully!\n");
}

// Display the schedule for a specific day
void display_day_schedule(int day) {
```

```
Enter your choice (1-6): 5
Enter the day number (1-7): 1

========= Day 1 Schedule =========
Subject 1: Physics          | Time: 12:00PM
===================================

===================================
       WEEKLY SCHEDULE MANAGER
===================================
1 - Add Subject to Day
2 - Edit Subject
3 - Delete Subject
4 - Display Entire Weekly Schedule
5 - Display Specific Day's Schedule
6 - Exit
===================================
Enter your choice (1-6):
```

4. Limitations

Fixed Limits for Days and Subjects: The program is limited to 7 days and 10 subjects per day. Users with more dynamic schedules may find this restrictive.

Lack of Time Conflict Check: The program does not verify if subjects overlap in time, potentially leading to scheduling errors.

Data Loss on Exit: There is no way to save or load schedules; all data is lost when the program is closed.

No Advanced Time Handling: The program only accepts basic time inputs (e.g., "10:30 AM") without validating or handling different formats.

No Reminders or Alerts: There is no reminder feature to alert users of upcoming classes, a critical feature for busy students.

Limited Error Handling: The program lacks sufficient error checks, such as for invalid subject names (e.g., names with spaces) or time input formats.

5. Future Scope

Dynamic Scheduling: Allow users to customize the number of days and subjects based on their needs, making the program more flexible.

Data Persistence: Implement file handling to save and load schedules, ensuring users don't lose their data upon program exit.

Time Conflict Check: Add a feature to detect time conflicts between subjects and prevent users from scheduling multiple classes at the same time.

Graphical User Interface (GUI): Transition from a text-based interface to a GUI, making the program more user-friendly and visually appealing.

Cloud Integration: Enable cloud syncing to access schedules from any device, making the program more accessible and convenient for users.

Search and Sorting: Implement features to search for subjects and sort them by time or name to make it easier to navigate large schedules.

Alarm/Reminder System: Add a reminder feature that alerts users 30 minutes before the start of a class. This could be implemented as a pop-up notification or an alarm.

Example Feature: The program could check the system time and compare it to the scheduled time for each subject. When the time difference is 30 minutes, a notification or alarm could be triggered, reminding the user of the upcoming class.

6. Inspired by:

Google Calendar: Google Calendar is a cloud-based application that provides advanced features such as cross-platform syncing, recurring events, and automatic reminders. Unlike the Weekly Schedule Manager, Google Calendar allows users to manage their schedules seamlessly across different devices and platforms.

Microsoft Outlook Calendar: Microsoft Outlook provides an integrated calendar with features like meeting scheduling, time zone handling, and reminders. It also integrates with other Microsoft Office tools, offering a more robust solution compared to the simple, text-based Weekly Schedule Manager.

Both Google Calendar and Microsoft Outlook offer advanced functionalities such as alarms, cross-platform syncing, and integration with other services (e.g., email), making them more comprehensive scheduling solutions than the Weekly Schedule Manager program.

TimeTree : A collaborative calendar app with a focus on sharing and coordination among groups. The premium version includes features like custom reminders and priority settings.

Calendly (Pro) : A scheduling tool designed mainly for meetings and appointments. The Pro version includes features like workflow automation, analytics, and custom branding.

7. Conclusion

The Weekly Schedule Manager is a basic yet functional tool for managing weekly schedules. It is especially useful for students who need a simple solution for organizing their subjects and times. However, it has several limitations, including fixed day and subject limits, lack of time conflict checking, and no reminder system. Future improvements such as data persistence, time conflict detection, and reminder features could greatly enhance the program’s usefulness. Despite these limitations, it serves as a great starting point for anyone learning C programming and looking to develop a scheduling tool.