Presented to the Gokongwei College of Engineering

De La Salle University - Manila

Term 2, A.Y. 2023-2024

Submitted by:

Miguel Carlos Estrada

Jannah Allysa Li

Viggo Aquino

Angellenne Pagcaliwagan

Submitted to:

Mr. Robert Jay Del Rosario

I. Daily Class Schedule Program

In the fast-paced world of education and work, efficient time management is crucial. As a group comprising Miguel Carlos D. Estrada, Jannah Allysa Li, Viggo Aquino, and Angellenne M. Pagcaliwagan, we recognize the need for a reliable tool that helps individuals organize their daily class schedules effectively. To address this, we have developed a user-friendly program that allows users to manage their classes seamlessly.

Our program aims to streamline the process of scheduling classes by providing essential features such as adding and removing classes, viewing the overall schedule, and ensuring that no time conflicts arise. Whether you're a student juggling multiple courses or a professional attending workshops and seminars, our program caters to your scheduling needs.

In this document, we will delve into the details of our program, including its objectives, problem statement, and the steps we took to create a robust solution. By adhering to the principles of S.M.A.R.T. objectives, we have designed a program that simplifies class management while maintaining flexibility and adaptability.

II. Objectives

Specificity (S):

- Develop a working program that allows users to manage their class schedules efficiently.
- Include features such as adding and removing classes, viewing the overall schedule, and handling potential conflicts.
- Design an intuitive interface that caters to users of varying technical backgrounds.

Measurability (M):

- Ensure that the program can handle up to 10 classes.
- Implement error checks for invalid inputs (e.g., class time outside the range of 0700 to 2100).
- Gather feedback from users to assess the program's effectiveness and usability.

Achievability (A):

- Utilize the C programming language to create the program.
- Work together effectively to achieve the desired functionality.
- Continuously test and refine the program during development.

Relevance (R):

- Provide a practical solution for individuals managing their daily class schedules.
- Encourage efficient time allocation for classes and other activities.
- Ensure the program can be customized for various users' needs.

Time-Bound (T):

- Aim to complete the program within the specified timeframe.
- Continuously improve the program based on user feedback and any identified issues.

III. Problem Statement

➢ The user has the option to:

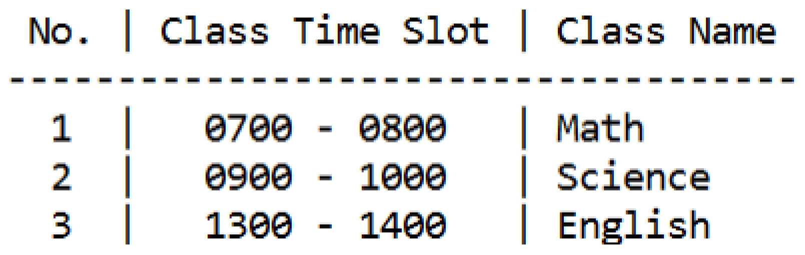
- Add Class
- Remove Class
- View Class Schedule
- Exit Program

➢ When adding a class:

- The program asks from the user:
    - The name of the class
    - What hour of the day the class begins
- It is assumed that all classes last for 1 hour
- The class will not be added if:
    - There are already 10 classes added
    - It conflicts with the schedule of other classes
    - It lies outside class hours   (0700 to 2100)

➢ When viewing the class schedule:

- ○ The details seen should be the same as in the example shown below:



```
 No. | Class Time Slot | Class Name
------------------------------------
  1  |   0700 - 0800   | Math
  2  |   0900 - 1000   | Science
  3  |   1300 - 1400   | English
```

- ○ The user cannot view the schedule if there are currently no classes added

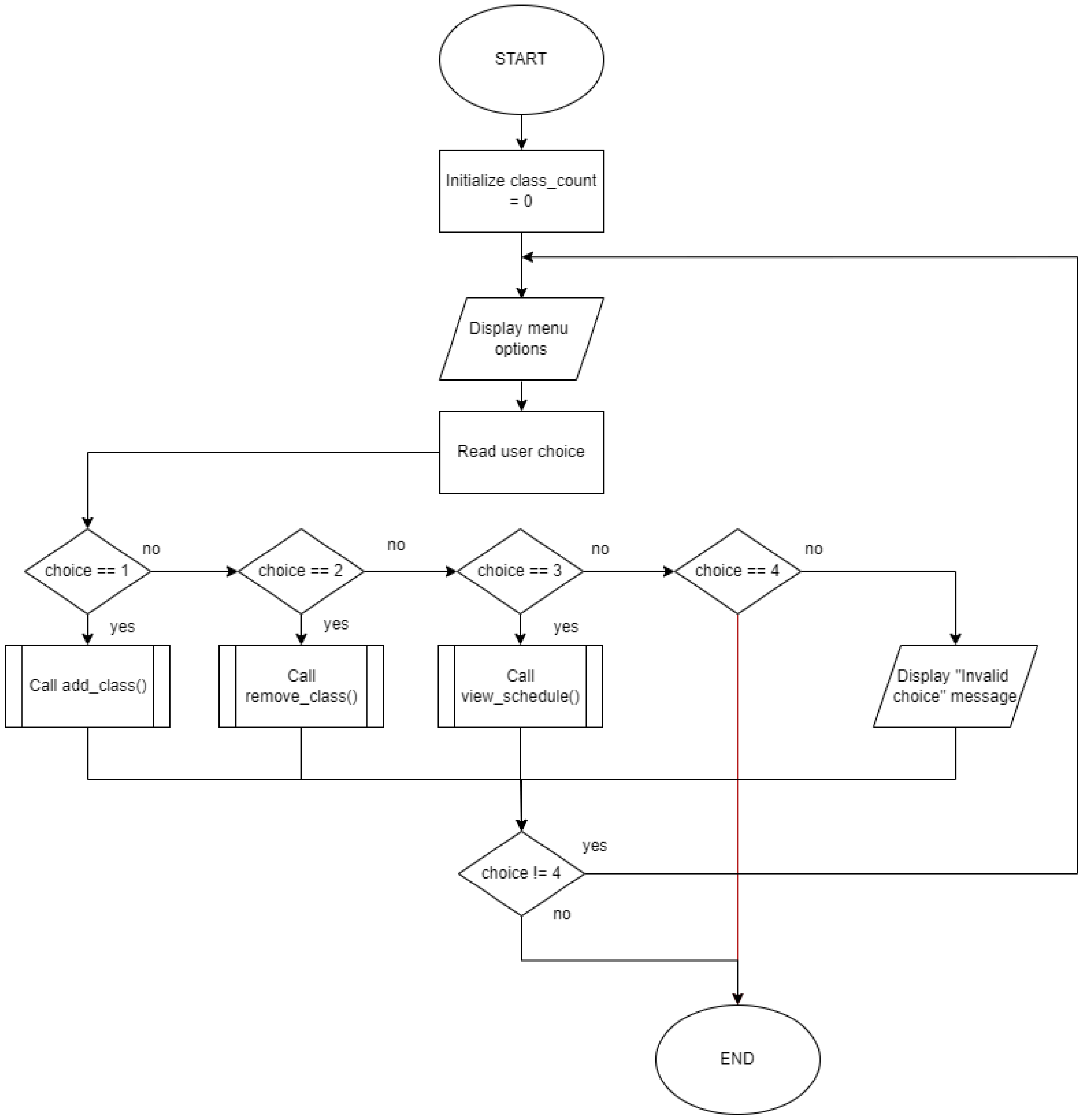
➢ When removing a class:

- The class schedule is viewed first
- Then, the user inputs the class no. corresponding to the class to be removed
- The other classes should adjust such that there is no gap in the numbering
    - (i.e. if #1 is removed, #2 becomes #1 and #3 becomes #2)
- ○ The user cannot remove any classesIf there are no more classes added

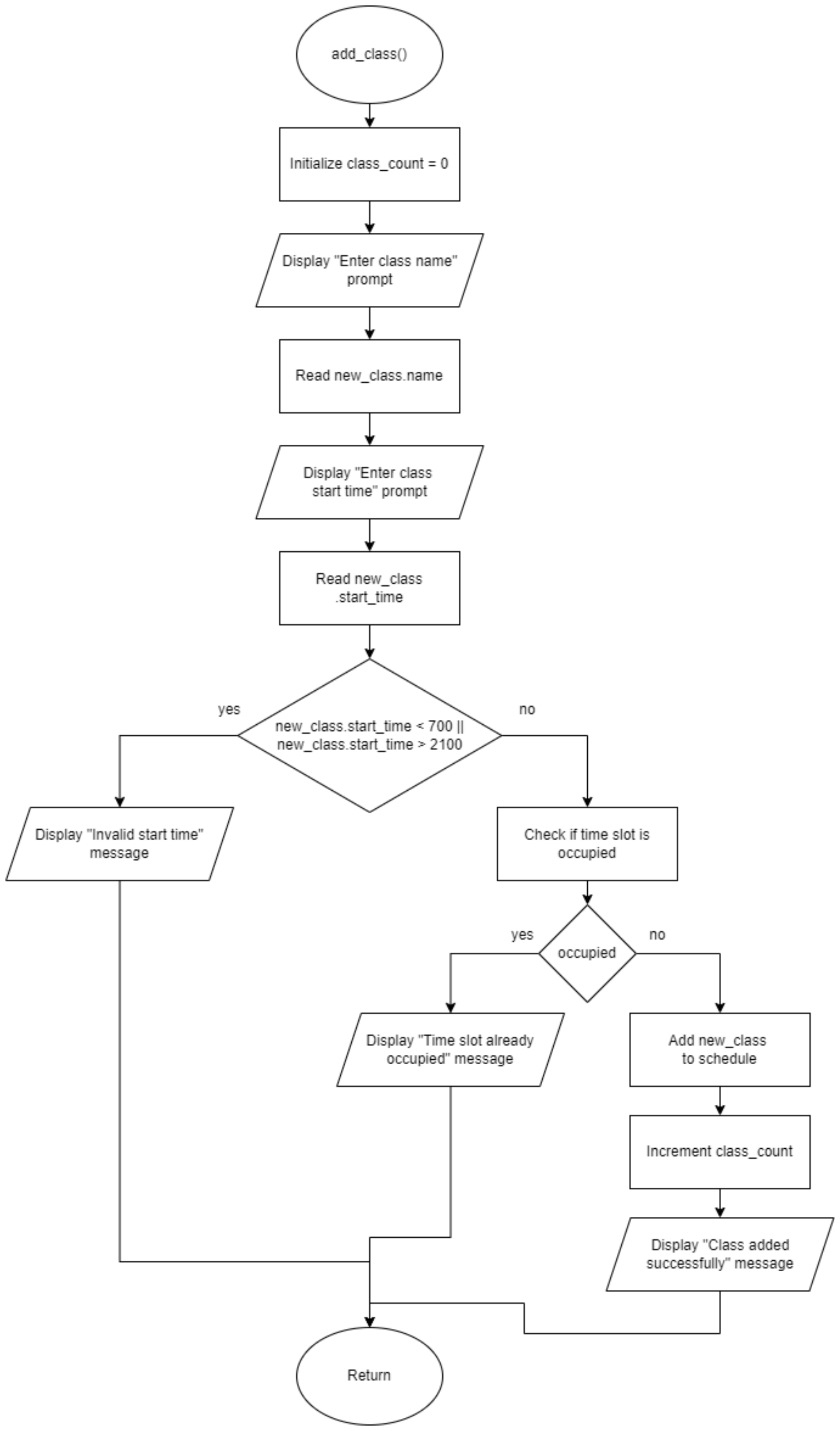
➢ Also, the program should be able to handle incorrect inputs(excluding inputs of a different data type)

IV. Flowchart

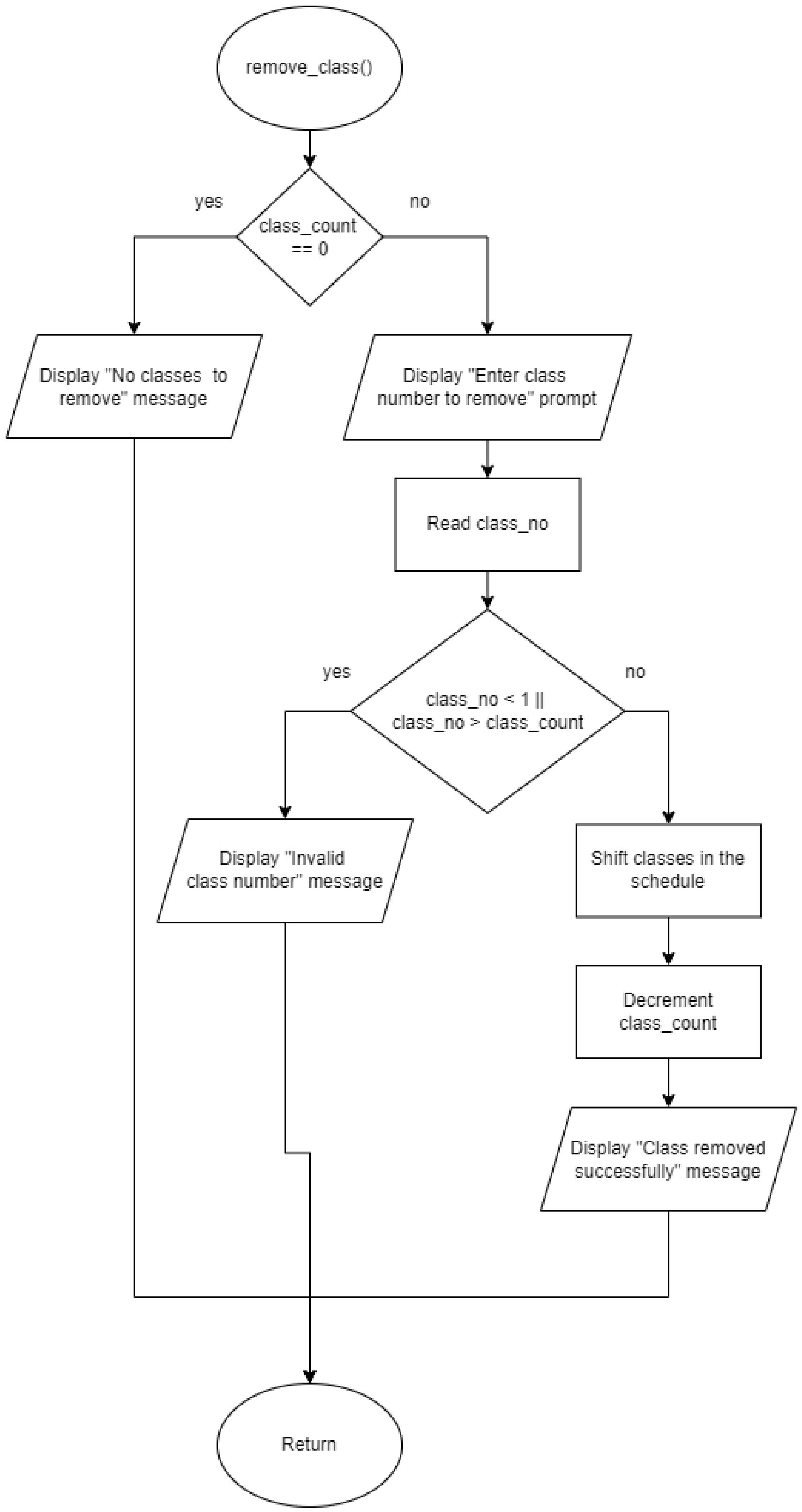
Main:



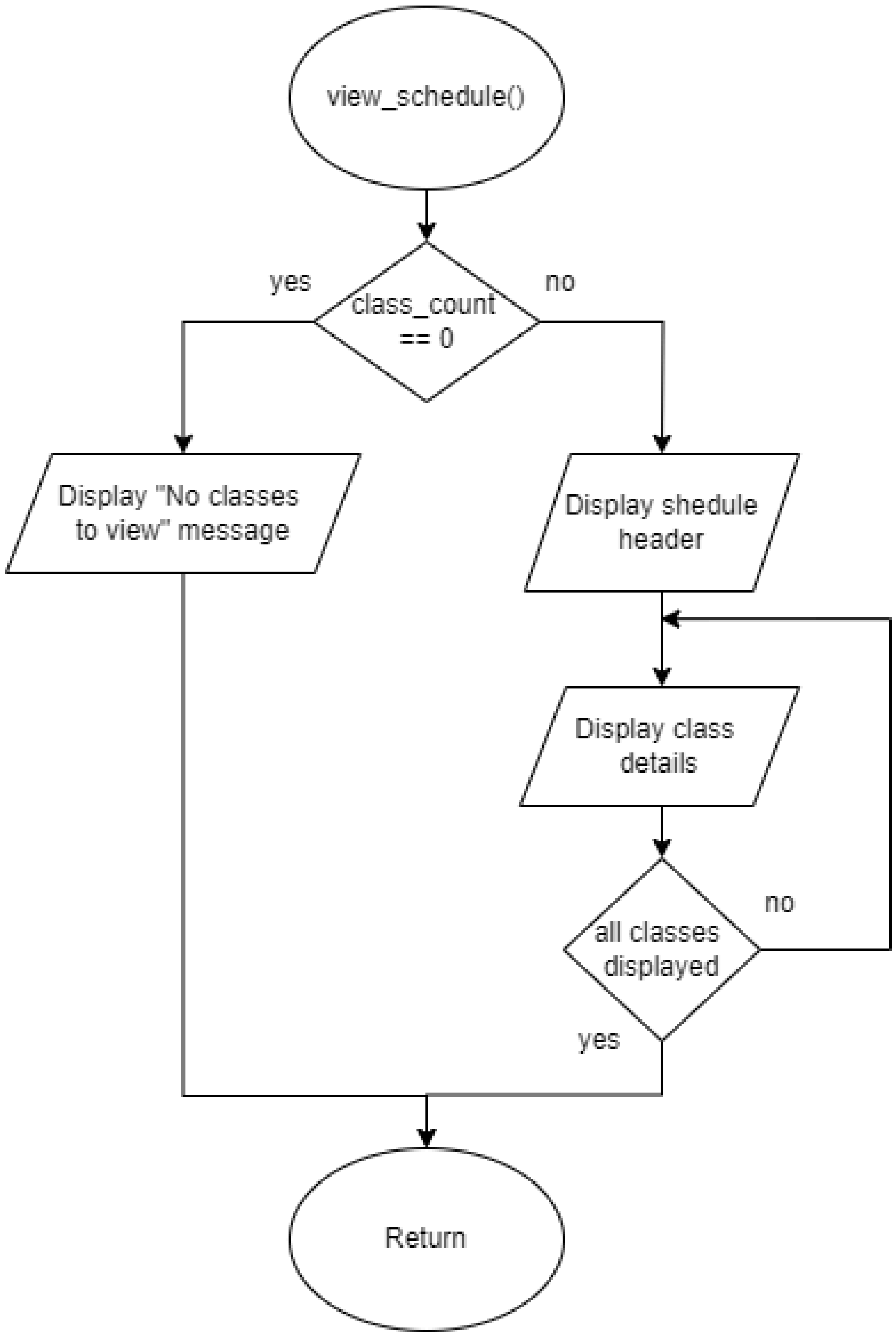
add_class Function:


add_class()
Initialize class_count = 0
Display "Enter class name" prompt
Read new_class.name
Display "Enter class start time" prompt
Read new_class .start_time
new_class.start_time < 700 || new_class.start_time > 2100
yes
no
Display "Invalid start time" message
Check if time slot is occupied
occupied
yes
no
Display "Time slot already occupied" message
Add new_class to schedule
Increment class_count
Display "Class added successfully" message
Return

remove_class Function:


remove_class()
yes
no
class_count == 0
Display "No classes to remove" message
Display "Enter class number to remove" prompt
Read class_no
yes
no
class_no < 1 || class_no > class_count
Display "Invalid class number" message
Shift classes in the schedule
Decrement class_count
Display "Class removed successfully" message
Return

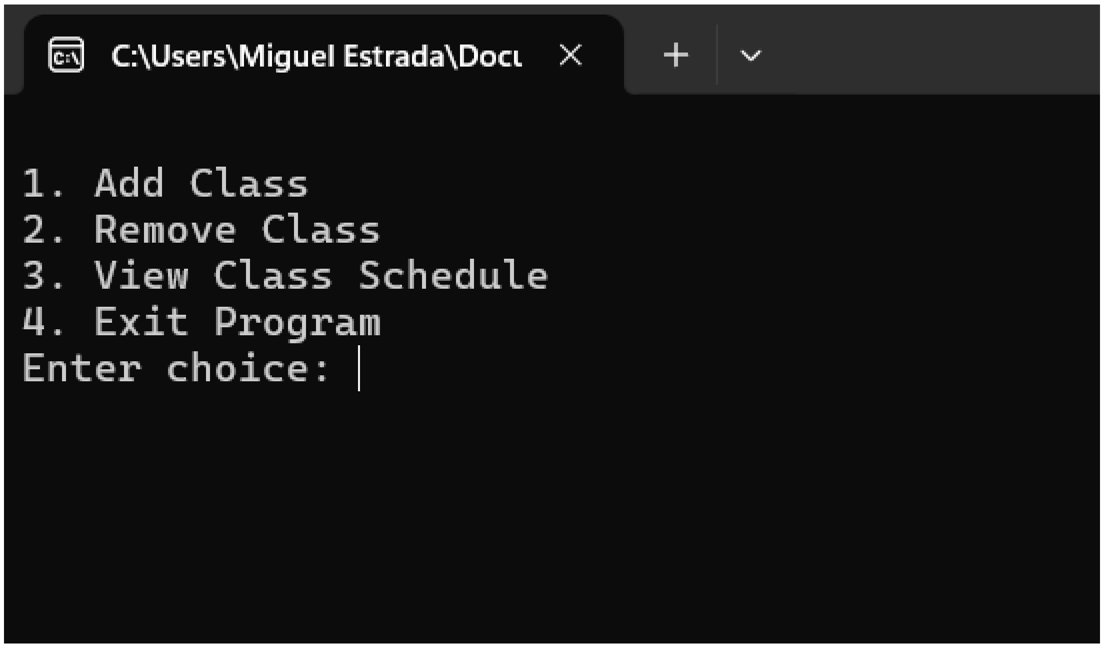
view_schedule Function:


view_schedule()
class_count == 0
yes
no
Display "No classes to view" message
Display shedule header
Display class details
all classes displayed
no
yes
Return

V. Codes

```
#include <stdio.h>
#include <string.h>

typedef struct {
    char name[50];
    int start_time;
} Class;

Class schedule[10];
int class_count = 0;

void add_class() {
    if (class_count >= 10) {
        printf("Cannot add more classes. Maximum of 10 classes reached.\n");
        return;
    }

    Class new_class;
    printf("Enter class name: ");
    scanf("%s", new_class.name);
    printf("Enter class start time (0700 to 2100): ");
    scanf("%d", &new_class.start_time);

    if (new_class.start_time < 700 || new_class.start_time > 2100) {
        printf("Invalid start time. Class not added.\n");
        return;
    }
    for (int i = 0; i < class_count; i++) {
        if (schedule[i].start_time == new_class.start_time) {
            printf("Time slot already occupied. Class not added.\n");
            return;
        }
    }

    schedule[class_count++] = new_class;
    printf("Class added successfully.\n");
}

void remove_class() {
    if (class_count == 0) {
        printf("No classes to remove.\n");
        return;
    }

    int class_no;
    printf("Enter class number to remove: ");
    scanf("%d", &class_no);

    if (class_no < 1 || class_no > class_count) {
        printf("Invalid class number.\n");
        return;
    }

    for (int i = class_no - 1; i < class_count - 1; i++) {
        schedule[i] = schedule[i + 1];
    }
    class_count--;
    printf("Class removed successfully.\n");
}

void view_schedule() {
    if (class_count == 0) {
        printf("No classes to view.\n");
        return;
    }

    printf("No. | Class Time Slot | Class Name\n");
    for (int i = 0; i < class_count; i++) {
         printf("%d   |   %04d - %04d   | %s\n", i + 1, schedule[i].start_time, schedule[i].start_time +
100, schedule[i].name);
    }
}

int main() {
    int choice;

    while (1) {
         printf("\n1. Add Class\n2. Remove Class\n3. View Class Schedule\n4. Exit Program\nEnter
choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                add_class();
                break;
            case 2:
                remove_class();
                break;
            case 3:
                view_schedule();
                break;
            case 4:
                return 0;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

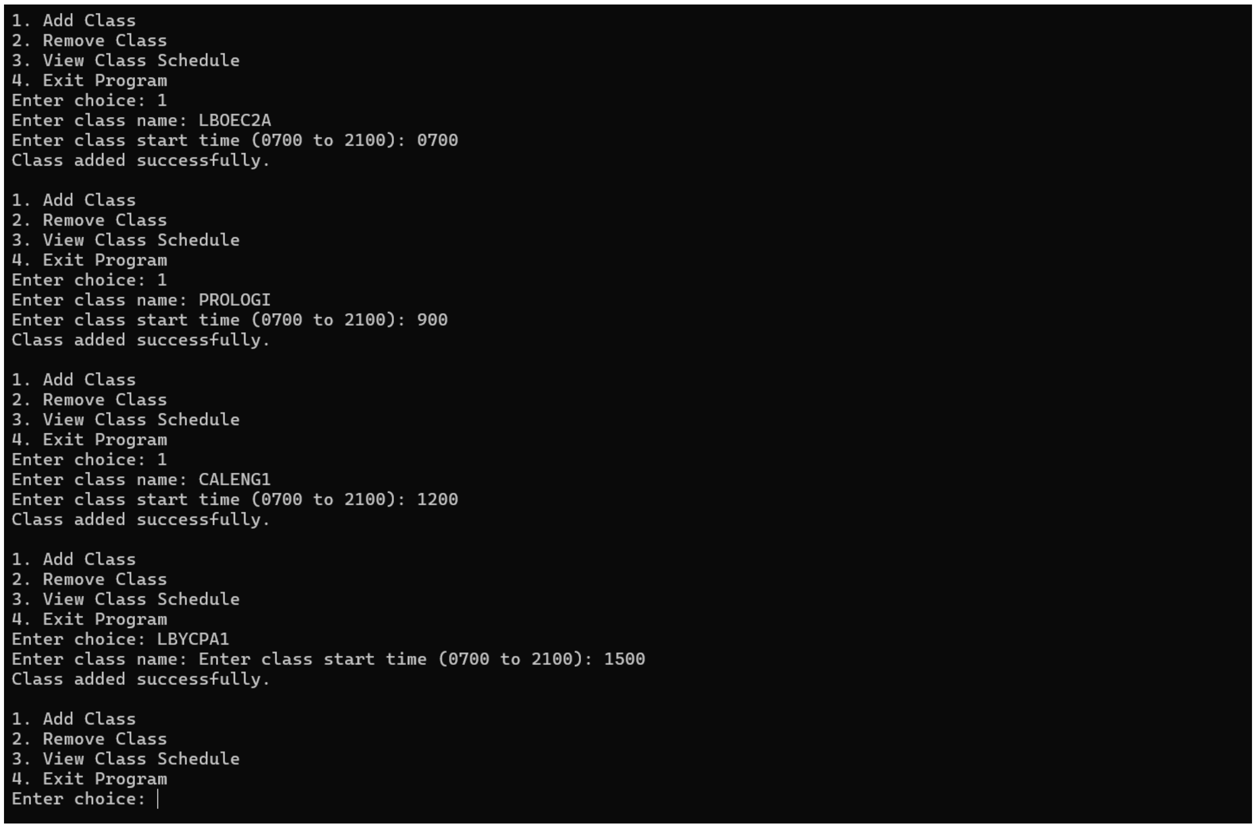
VI. Output Screenshots



C:\Users\Miguel Estrada\Docu

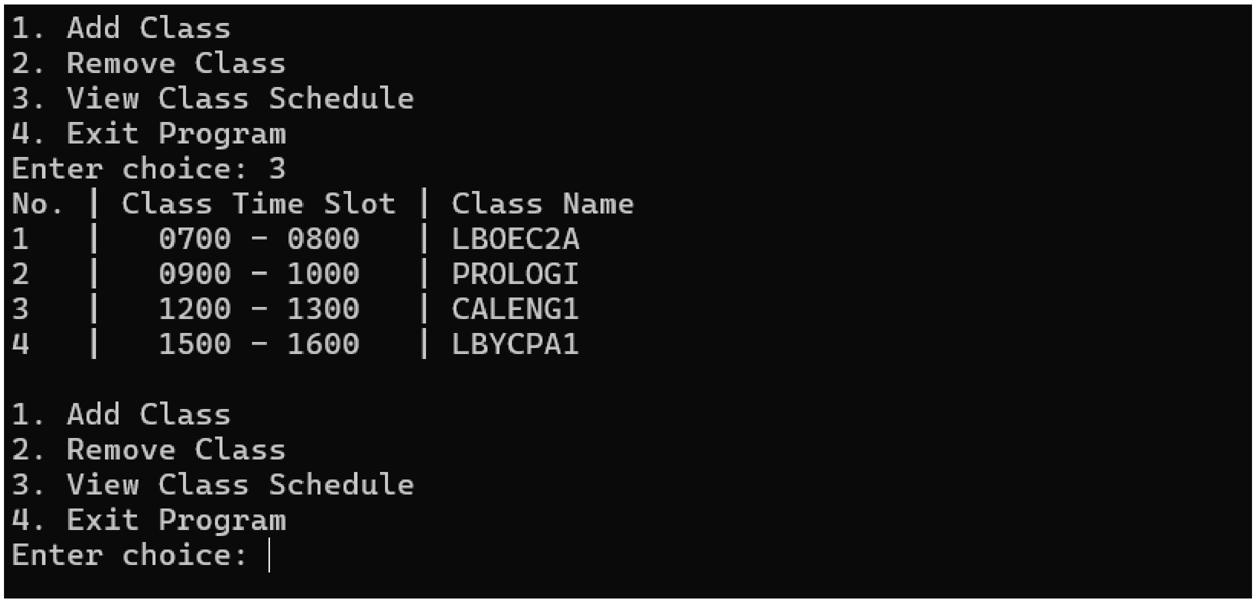
```
1. Add Class
2. Remove Class
3. View Class Schedule
4. Exit Program
Enter choice: 
```

Adding Classes:

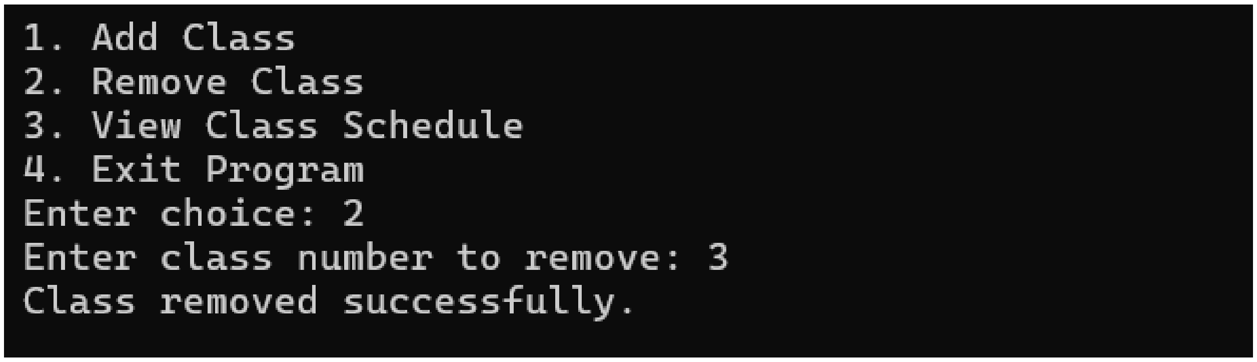


```
1. Add Class
2. Remove Class
3. View Class Schedule
4. Exit Program
Enter choice: 1
Enter class name: LBOEC2A
Enter class start time (0700 to 2100): 0700
Class added successfully.

1. Add Class
2. Remove Class
3. View Class Schedule
4. Exit Program
Enter choice: 1
Enter class name: PROLOGI
Enter class start time (0700 to 2100): 900
Class added successfully.

1. Add Class
2. Remove Class
3. View Class Schedule
4. Exit Program
Enter choice: 1
Enter class name: CALENG1
Enter class start time (0700 to 2100): 1200
Class added successfully.

1. Add Class
2. Remove Class
3. View Class Schedule
4. Exit Program
Enter choice: LBYCPA1
Enter class name: Enter class start time (0700 to 2100): 1500
Class added successfully.

1. Add Class
2. Remove Class
3. View Class Schedule
4. Exit Program
Enter choice: 
```

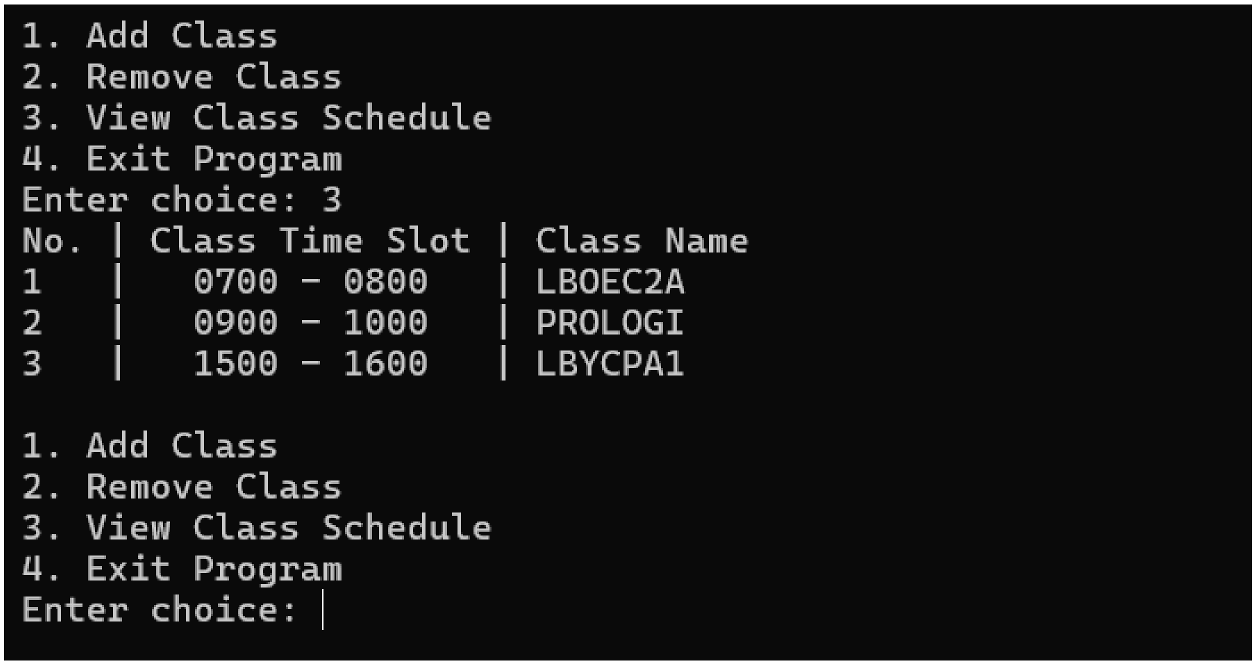
Viewing current class schedule:


```
1. Add Class
2. Remove Class
3. View Class Schedule
4. Exit Program
Enter choice: 3
No. | Class Time Slot | Class Name
1   |   0700 - 0800   | LBOEC2A
2   |   0900 - 1000   | PROLOGI
3   |   1200 - 1300   | CALENG1
4   |   1500 - 1600   | LBYCPA1

1. Add Class
2. Remove Class
3. View Class Schedule
4. Exit Program
Enter choice: 
```

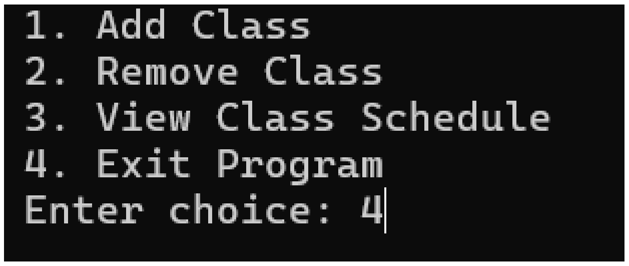
Removing Class (CALENG1):


```
1. Add Class
2. Remove Class
3. View Class Schedule
4. Exit Program
Enter choice: 2
Enter class number to remove: 3
Class removed successfully.
```

Viewing current class schedule:


```
1. Add Class
2. Remove Class
3. View Class Schedule
4. Exit Program
Enter choice: 3
No. | Class Time Slot | Class Name
1   |   0700 - 0800   | LBOEC2A
2   |   0900 - 1000   | PROLOGI
3   |   1500 - 1600   | LBYCPA1

1. Add Class
2. Remove Class
3. View Class Schedule
4. Exit Program
Enter choice: 
```

Exit Program:


```
1. Add Class
2. Remove Class
3. View Class Schedule
4. Exit Program
Enter choice: 4
```

VII. Conclusion

In our journey to create the Daily Class Schedule Program, our group has navigated the intricacies of software development with a clear vision: to simplify class management for users. As we conclude this document, let us revisit the key points, celebrate our achievements, and consider the broader implications of our work.

Our program addresses the pressing need for efficient time management in education and work. By allowing users to add, remove, and view their class schedules seamlessly, we empower them to allocate their time effectively. The S.M.A.R.T. objectives guided our development process, ensuring that our solution remains practical, measurable, and adaptable.

Our primary goal was to create a user-friendly program that streamlines class scheduling. Whether you're a student juggling multiple courses or a professional attending workshops, our program caters to your needs. By adhering to the principles of simplicity and functionality, we've crafted an intuitive interface that transcends technical barriers.

And throughout this journey, we've achieved several milestones:

- **Functionality:** Our program allows users to add classes, handle conflicts, and view their schedules effortlessly.
- **Error Handling:** We implemented robust checks to prevent invalid inputs, ensuring a smooth user experience.
- **Collaboration:** As a team, we worked harmoniously, combining our skills to create a cohesive solution.

Addressing Counter Arguments:

While we focused on efficiency and simplicity, alternative approaches exist. Some might argue for additional features or more elaborate error messages. Acknowledging these perspectives underscores our commitment to thoroughness.

Considering the Bigger Picture:

Our program is a small piece of the software ecosystem, yet it contributes significantly. By enabling effective time management, we empower users to excel in their academic and professional pursuits. As developers, we recognize our role in shaping tools that impact people's lives.

In conclusion, the Daily Class Schedule Program isn't just lines of code; it's a solution that empowers individuals one class at a time. As we continue refining and gathering feedback, we remain committed to enhancing the lives of our users through thoughtful software design.