



Project Title

**STUDENT MANAGEMENT SYSTEM**

(MANOJ KUMAR D) (2347237)

Submitted to

Dr.B.J.Hubert Shanthan

7<sup>th</sup> October 2023

## TABLE OF CONTENTS

| S.No | Content                     | Page No |
|------|-----------------------------|---------|
| 1    | Introduction                | 3       |
| 2    | Functionalities and Modules | 4-7     |
| 3    | Source Code                 | 8-26    |
| 4    | Screen Shots                | 27-36   |
| 5    | Conclusion                  | 37      |

# INTRODUCTION

In today's rapidly evolving educational landscape, the Student Information Management System (SIMS) stands as a pivotal tool, reshaping how institutions handle student data. Our SIMS solution is meticulously crafted to simplify the complex task of managing student information, ensuring accuracy, security, and efficiency.

Designed with the needs of educational institutions in mind, our SIMS enables seamless storage of vital student details such as first name, last name, unique roll number, GPA, and registered courses. This user-friendly system serves as the digital repository, fostering streamlined communication and data-driven decision-making among administrators, teachers, and staff.

By centralizing student data, our SIMS facilitates informed decision-making processes, enhances administrative efficiency, and provides a comprehensive overview of student performance. It empowers institutions to focus on delivering high-quality education, fostering a conducive learning environment, and nurturing individual student growth.

Incorporating advanced security measures, our SIMS ensures the confidentiality and integrity of student data, adhering to the highest standards of data protection. As educational institutions embrace the digital era, our SIMS emerges as the cornerstone for organized, efficient, and proactive management of student information, laying the foundation for a brighter future in education.

## **Problem Statement**

A simple software for student information management system which can perform the following operations:

- 1.Store first name of the student.
- 2.Store last name of the student.
- 3.Store unique roll number for every student.
- 4.Store GPA for every student.
- 5.Store courses registered by the student.

# FUNCTIONALITIES AND MODULES

## **Functionality/Approach**

The idea is to form an individual functions for every operation. All the functions are unified to form software.

- 1.Add student details from file.
- 2.Add student details manually.
- 3.Find the student by the given roll number.
- 4.Find the student by the given first name.
- 5.Find the student registered in a course.
- 6.Count number of students.
- 7.Delete a student by the given roll number.
- 8.Update a student by the given roll number.
- 9.Print all student's data.
- 10.Exit the program and save all data in file to recover back.

## **Idea**

The software will consist of 5 files main.c, Student\_sys.c, Student\_sys.h, updated\_students.txt, and Students.txt.

The main.c will contain the interface and calling the function, Student\_sys.c will contain the global variables and the body of the functions, Student\_sys.h will contain the definitions and the prototypes, updated\_students.txt will contain the data before exit, students.txt will contain the data.

- 1.We will have array of struct of 50 elements which is the max number of students.
- 2.Each struct contains the student details.
- 3.We will have a global index which refers to the number of students.

4.The index is initialized with zero and can go up to 50.

## **Modules:**

### **Main.c**

The main.c is just a simple file which consist of:

- Infinite loop while until the choice = 10.
- Printf and scanf functions.
- Calling the program functions through switch cases.

### **Student\_sys.h**

The Student\_sys.h consist of:

- Macro to make things easier.
- Struct student\_info which contains the details of a student.
- Type definition of struct student info to use it easier with “Item”.
- Type definition of FIFO Buffer.
- Function prototypes with their usage.

### **Student\_sys.c**

The Student\_sys.c consist of:

- FILE \*students\_file, \*updated\_students\_file which is a pointer to a typedef FILE to manage I/O file.
- Private APIs for this file only.
- Functions implementations.

#### **1- students\_sys\_init**

- The function will start by checking the input parameters validity.
- FIFO (Queue) Initialization.
- Printing if passed.

## **2- add\_student\_manually**

- The function will start ask the roll number.
- Scan if the roll number is existed.
- If not, continue reading data form the user.
- After getting all data add it to the buffer and print information.

## **3- add\_student\_from\_file**

- This function starts with open the student.txt file with option read only.
- Check if the file is exited and there is data?!
- Put the file into a while loop until reach the last line in the file.
- Start to reading data from the student.txt file:
  - 1-Roll number and search if it exists.
  - 2-First name.
  - 3-Last name.
  - 4-GPA Score.
  - 5-5 Course IDs sequentially.
- Printing info if everything is correctly.

## **4- find\_student\_by\_roll**

- This function it's about searching for the student in the queue with roll number.
- Starting with check if there is students in the queue?
- Asking him for the roll number and show all details.

## **5- find\_student\_by\_firstname**

- This function it's about searching for the student in the queue with first name.
- Starting with check if there is students in the queue?

- Asking him for the first number and show all details.

#### **6- find\_student\_by\_course**

- This function it's about searching for the student in the queue with course ID.
- Starting with check if there is students in the queue?
- Asking him for the ID and show all details.
- Telling the number of students is enrolled.

#### **7- print\_students\_count**

- Printing the Total number of students.
- Printing the remaining of student which can enter.

#### **8- delete\_student\_by\_roll**

- Start with checking there is students in the queue.
- Asking to enter the roll number to remove.
- Start removing progress and show information.

#### **9- update\_student\_by\_roll**

- Start with checking there is students in the queue.
- Asking to enter the roll number to update.
- Start updating progress:
  - Roll number and search if it exists.
  - First name.
  - Last name.
  - GPA Score.
  - Course IDs and ask which one using switch case.
  -

#### **10- update\_student\_file**

- This function for save all students data in file before exit form the program.

# SOURCE CODE

## Student\_Sys.c

```
#include "Student_Sys.h"

FILE *students_file, *updated_students_file;

// Private APIs
/* we put the declaration here because we need it as static in this file only*/

static void print_student_info(struct student_info *student);

static struct student_info *search_student_by_roll(FIFO_Buf_st *students_queue, int roll);

// Student Queue Initialization
FIFO_Status_st students_sys_init(FIFO_Buf_st *students_queue, Item *item, int length)
{
    // Check parameters validity
    if(!students_queue || !item || !length)
    {
        printf("Student System Queue Initialization Failed\n");
        return FIFO_NULL;
    }

    // FIFO Initialization process
    students_queue->base = item;
    students_queue->head = students_queue->base;
    students_queue->tail = students_queue->base;
    students_queue->length = length;
    students_queue->counter = 0;

    printf("Student System Queue Initialization Passed\n");
    return FIFO_NO_ERROR;
}

// Enter student data form file
void add_student_from_file(FIFO_Buf_st *students_queue)
{
    Item new_student;
    int i;

    // Opening a student.txt file with option read
    students_file = fopen("students.txt", "r");

    // Check if the file exist and there is data or not
    if(students_file == NULL)
    {
        printf("\n [ERROR] student.txt file not found. \n");
    }
}
```



```

printf("\n [ERROR] adding students from file failed. \n");
return;
}

// Reading students until end of file this using of feof == Check end of file
while(!feof(students_file))
{
    // Reading roll number of the student
    fscanf(students_file, "%d", &new_student.roll_number);

    // Check if the roll number is exists
    if(search_student_by_roll(students_queue, new_student.roll_number) != NULL)
    {
        // Printing that we find a number with other student
        printf("\n[ERROR] Roll number %d is already taken\n", new_student.roll_number);

        // Ignore the rest of the line
        fscanf(students_file, "%*[^\\n]");

        // Start over form next line in text file
        continue;
    }

    // Reading data first name, last name and GPA sequential
    fscanf(students_file, "%s", new_student.first_name);
    fscanf(students_file, "%s", new_student.last_name);
    fscanf(students_file, "%f", &new_student.GPA);

    // Reading course IDs
    for (i = 0; i < COURSES_NUMBER; ++i)
    {
        fscanf(students_file, "%d", &new_student.course_id[i]);
    }

    // Enqueue new student
    if((FIFO_enqueue(students_queue, new_student))== FIFO_NO_ERROR)
    {
        printf("\n[INFO] Roll number %d is saved successfully\n", new_student.roll_number);
    }
    else
    {
        printf("\n[ERROR] Adding students by file failed\n");
        return;
    }
}

printf("\n[INFO] Students details are saved successfully\n");

```

```

// Closing the file
fclose(students_file);
}

// Enter student data manually from console
void add_student_manually(FIFO_Buf_st *students_queue)
{
    Item new_student;
    int i;

    printf("\n=== Enter student data ===\n");
    printf("\tEnter roll number: ");
    scanf("%d", &new_student.roll_number);

    // Scan if the roll number which entered is exist
    if(search_student_by_roll(students_queue, new_student.roll_number))
    {
        printf("\n[ERROR] Roll number %d is already taken\n", new_student.roll_number);
        printf("\n[ERROR] Adding student manually failed\n");
        return;
    }

    // If not, Continue reading other data
    printf("\tEnter first name: ");
    scanf("%s", new_student.first_name);

    printf("\tEnter second name: ");
    scanf("%s", new_student.last_name);

    printf("\tEnter GPA: ");
    scanf("%f", &new_student.GPA);

    printf("\tEnter Courses IDs\n");
    for (i = 0; i < COURSES_NUMBER; ++i)
    {
        printf("\t\tEnter Courses no.%d: ", i+1);
        scanf("%d", &new_student.course_id[i]);
    }

    // Add new student
    if(FIFO_enqueue(students_queue, new_student) == FIFO_NO_ERROR)
    {
        printf("\n[INFO] Student details us saved successfully\n");
    }
    else
    {
        printf("\n[ERROR] Adding students manually failed\n");
        return;
    }
}

```

```

    }
}

// Get student date by its roll number
void find_student_by_roll(FIFO_Buf_st *students_queue)
{
    int input_number;
    Item *student;
    FIFO_Status_st queue_status;

    // Checking if the queue is empty
    queue_status = FIFO_is_empty(students_queue);
    if((queue_status == FIFO_EMPTY) || (queue_status == FIFO_NULL))
    {
        printf("\n[ERROR] Find student by roll number failed\n");
        return;
    }

    // Enter roll number you want to find
    printf("\nEnter roll number: ");
    scanf("%d", &input_number);

    // Scan in the queue to find this roll number
    student = search_student_by_roll(students_queue, input_number);

    // Check if we find roll number in the queue
    if (student == NULL)
    {
        printf("\n[ERROR] Roll number %d in not found\n", input_number);
        return;
    }
    else
    {
        // If we found the roll number lets print all data
        print_student_info(student);
    }
}

// Get student date by its first name
void find_student_by_firstname(FIFO_Buf_st *students_queue)
{
    Item *student = students_queue->tail;
    char input_name[NAME_LENGTH], i;

    FIFO_Status_st queue_status;

    // Checking if the queue is empty
    queue_status = FIFO_is_empty(students_queue);

```

```

if((queue_status == FIFO_EMPTY) || (queue_status == FIFO_NULL))
{
    printf("\n[ERROR] Find student by first name failed\n");
    return;
}

printf("\nEnter First Name: ");
scanf("%s",input_name);

// Loop inside queue
for (i = 0; i < students_queue->counter; ++i)
{
    // Compare between two string the input name and the name in queue to search about it
    if (!(strcmp(input_name, student->first_name)))
    {
        print_student_info(student);
        return;
    }
    else
    {
        printf("\n[ERROR] %s in not found\n",input_name);
        return;
    }

    // Check if we reach the last item in the queue
    if((student +1) == (students_queue->base + students_queue->length))
    {
        // Set to the start
        student = students_queue->base;
    }
    else
    {
        // Just go to the next tail :D
        student++;
    }
}

// Get student date by its course
void find_student_by_course(FIFO_Buf_st *students_queue)
{
    Item *student;
    int input_id, i, j, number_enroled_student = 0;

    FIFO_Status_st queue_status;

    // Checking if the queue is empty

```

```

queue_status = FIFO_is_empty(students_queue);

if((queue_status == FIFO_EMPTY) || (queue_status == FIFO_NULL))
{
    printf("\n[ERROR] Find student by course failed\n");
    return;
}

// Get the ID want to search about
printf("\nEnter Course ID: ");
scanf("%d", &input_id);

// Loop inside queue
student = students_queue->tail;
for (i = 0; i < students_queue->counter; ++i)
{
    // Scan inside every item
    for(j = 0; j < COURSES_NUMBER; ++j)
    {
        // Check if we find student have this ID
        if(student->course_id[j] == input_id)
        {
            print_student_info(student);
            number_enroled_student++;
            printf("\n");
            break;
        }
    }

    // Check if we reach the last item in the queue
    if((student + 1) == (students_queue->base + students_queue->length))
    {
        // Set to the start
        student = students_queue->base;
    }
    else
    {
        // Just go to the next tail :D
        student++;
    }
}

// Check if not found in any student
if(number_enroled_student == 0)
{
    printf("\n[ERROR] Course id %d is not found\n", input_id);
}
else

```

```

    {
        printf("\n[INFO] Total number of students enrolled: %d\n", number_enroled_student);
    }
}

// Get students number in queue
void print_students_count(FIFO_Buf_st *students_queue)
{
    int counter = students_queue->counter;
    int capacity = students_queue->length;

    printf("[INFO] Total number of students is %d\n", counter);
    printf("[INFO] You can add up to %d students\n", capacity);
    printf("[INFO] You can add %d more students\n", capacity - counter);
}

// Delete student from the queue
void delete_student_by_roll(FIFO_Buf_st *students_queue)
{
    int input_number, i, flag = 0;
    Item *student;

    FIFO_Status_st queue_status;

    // Checking if the queue is empty
    queue_status = FIFO_is_empty(students_queue);

    if((queue_status == FIFO_EMPTY) || (queue_status == FIFO_NULL))
    {
        printf("\n[ERROR] Delete student by roll number failed\n");
        return;
    }

    // Enter roll number you want to delete
    printf("\nEnter roll number: ");
    scanf("%d", &input_number);

    student = students_queue->base;
    // Loop inside queue
    for (i = 0; i < students_queue->counter; ++i)
    {
        if(student->roll_number == input_number)
        {
            // Deleting student
            *student = *(students_queue->tail);

            // Update counter of queue

```

```

students_queue->counter--;

// Check if we reach the last item in the queue
if((students_queue->tail + 1) == (students_queue->base + students_queue->length))
{
    // Set to the start
    students_queue->tail = students_queue->base;
}
else
{
    // Just go to the next tail :D
    students_queue->tail++;
}

flag = 1;
break;
}
else
{
    flag = 0;
}
student++;
}

if(flag == 1)
{
    printf("\n[INFO] The Roll Number is removed successfully\n");
}
else
{
    printf("\n[ERROR] This Roll Number %d not found\n",input_number);
}
}

// Update a specific data in queue
void update_student_by_roll(FIFO_Buf_st *students_queue)
{
    Item *update_student, *student;
    int i, input_roll , input_option, input_new_roll, flag = 0;

    FIFO_Status_st queue_status;

    // Checking if the queue is empty
    queue_status = FIFO_is_empty(students_queue);

    if((queue_status == FIFO_EMPTY) || (queue_status == FIFO_NULL))
    {
        printf("\n[ERROR] Delete student by roll number failed\n");
    }
}

```

```

    return;
}

// Enter roll number you want to update
printf("\nEnter roll number: ");
scanf("%d", &input_roll);

// Search about the input roll to get its whole item
update_student = search_student_by_roll(students_queue, input_roll);

// Check if we find roll number in the queue
if (update_student == NULL)
{
    printf("\n[ERROR] Roll number %d in not found\n", input_roll);
    return;
}
else
{
    // If we found the roll number lets print all data
    printf("\n==== Student data ==== \n");
    print_student_info(update_student);
}

printf("\nWhich date do you want to change ?\n");
printf("\t 1: The Roll Number\n");
printf("\t 2: The First Name\n");
printf("\t 3: The Second Name\n");
printf("\t 4: The GPA Score\n");
printf("\t 5: The Courses ID\n");
printf("Enter your option: ");

// Get the option
scanf("%d",&input_option);
switch (input_option)
{
    case 1:
        printf("Enter the new roll number: ");
        scanf("%d",&input_new_roll);

        student = students_queue->tail;
        // Loop inside the queue
        for (i = 0; i < students_queue->counter; ++i)
        {
            // Check if we find the the roll we search about to break
            if (student->roll_number == input_new_roll)
            {
                // Get out form for loop
                printf("\n[ERROR] This Roll Number %d is exist\n",input_new_roll);
            }
        }
    }
}

```



```

        // If we find a roll number match set flag
        flag = 1;
        break;
    }

    // Check if we reach the last item in the queue
    if((student + 1) == (students_queue->base + students_queue->length))
    {
        student = students_queue->base;
    }
    else
    {
        // Just go to the next tail :D
        student++;
    }
}

// If the flag doesn't changed this mean that no roll number match
if(flag == 0)
{
    update_student->roll_number = input_new_roll;
    printf("\n[INFO] The Roll Number %d in updated successfully\n", input_new_roll);
}
break;

case 2:
    printf("Enter the new first name: ");
    scanf("%s", update_student->first_name);
    printf("\n[INFO] The First Name %s in updated successfully\n", update_student->first_name);
    break;

case 3:
    printf("Enter the new last name: ");
    scanf("%s", update_student->last_name);
    printf("\n[INFO] The Last Name %s in updated successfully\n", update_student->last_name);
    break;

case 4:
    printf("Enter the new GPA: ");
    scanf("%f", &update_student->GPA);
    printf("\n[INFO] The GPA Score %0.1f in updated successfully\n", update_student->GPA);
    break;

case 5:
    printf("Enter the course number from %d to %d: ", 1, COURSES_NUMBER);
    scanf("%d", &input_option);
    printf("Enter the new course id: ");

```

```

        scanf("%d", &update_student->course_id[input_option - 1]);
        printf("\n[INFO] The Course ID %d in updated successfully\n", update_student-
>course_id[input_option - 1]);
        break;

    default:
        break;
}
}

// Update students and what we do in file
void update_student_file(FIFO_Buf_st *students_queue)
{
    Item *student;
    int i, j;

    // Opening a update_students.txt file with w option to write in it
    updated_students_file = fopen("update_students.txt", "w");

    // Check if the file exist and there is data or not
    if(students_file == NULL)
    {
        printf("\n [ERROR] update_students.txt create failed. \n");
        printf("\n [ERROR] update students file failed. \n");
        return;
    }

    student = students_queue->tail;
    // Loop inside the queue
    for (i = 0; i < students_queue->counter; ++i)
    {
        // Write into file
        fprintf(updated_students_file, "%d ", student->roll_number);
        fprintf(updated_students_file, "%s ", student->first_name);
        fprintf(updated_students_file, "%s ", student->last_name);
        fprintf(updated_students_file, "%0.1f ", student->GPA);

        for(j = 0; j < COURSES_NUMBER ; ++j)
        {
            fprintf(updated_students_file, "%d ", student->course_id[j]);
        }

        // New line after every Item
        fprintf(updated_students_file, "\n");

        // Check if we reach the last item in the queue
        if((student + 1) == (students_queue->base + students_queue->length))
        {

```

```

        student = students_queue->base;
    }
    else
    {
        // Just go to the next tail :D
        student++;
    }

}

printf("\n[INFO] Update Students details are saved in file successfully\n");

// Closing the file
fclose(updated_students_file);
}

// Enter student data form update file
void add_student_from_update_file(FIFO_Buf_st *students_queue)
{
    Item new_student;
    int i;

    // Opening a update_students.txt file with option read
    updated_students_file = fopen("update_students.txt", "r");

    // Check if the file exist and there is data or not
    if(updated_students_file == NULL)
    {
        printf("\n [ERROR] updated_students_file.txt file not found. \n");
        printf("\n [ERROR] adding students from update file failed. \n");
        return;
    }

    // Reading students until end of file this using of feof == Check end of file
    while(!feof(updated_students_file))
    {
        // Reading roll number of the student
        fscanf(updated_students_file, "%d", &new_student.roll_number);

        // Check if the roll number is exists
        if(search_student_by_roll(students_queue, new_student.roll_number) != NULL)
        {
            // Printing that we find a number with other student
            printf("\n[ERROR] Roll number %d is already taken\n", new_student.roll_number);

            // Ignore the rest of the line
            fscanf(updated_students_file, "%*[^\\n]");

            // Start over from next line in text file

```

```

        continue;
    }

    // Reading data first name, last name and GPA sequential
    fscanf(updated_students_file, "%s", new_student.first_name);
    fscanf(updated_students_file, "%s", new_student.last_name);
    fscanf(updated_students_file, "%f", &new_student.GPA);

    // Reading course IDs
    for (i = 0; i < COURSES_NUMBER; ++i)
    {
        fscanf(updated_students_file, "%d", &new_student.course_id[i]);
    }

    // Enqueue new student
    if((FIFO_enqueue(students_queue, new_student)) == FIFO_NO_ERROR)
    {
        printf("\n[INFO] Roll number %d is saved successfully\n", new_student.roll_number);
    }
    else
    {
        printf("\n[ERROR] Adding students by update file failed\n");
        return;
    }
}

printf("\n[INFO] Students Recover are saved successfully\n");

// Closing the file
fclose(updated_students_file);
}

// Print all students in the queue
void show_students_info(FIFO_Buf_st *students_queue)
{
    Item *student;
    char i;
    FIFO_Status_st queue_status;

    // Checking if the queue is empty
    queue_status = FIFO_is_empty(students_queue);
    if((queue_status == FIFO_EMPTY) || (queue_status == FIFO_NULL))
    {
        printf("\n[ERROR] Show students info failed\n");
        return;
    }

    printf("\nAll Students\n\n");

```

```

student = students_queue->tail;
for (i = 0; i < students_queue->counter; ++i)
{
    print_student_info(student);
    printf("\n");

    // Check if we reach the last item in the queue
    if((student + 1) == (students_queue->base + students_queue->length))
    {
        student = students_queue->base;
    }
    else
    {
        // Just go to the next tail :D
        student++;
    }
}

static void print_student_info(struct student_info *student)
{
    int i;

    printf("The student details are\n");
    printf("\tFirst Name: %s\n", student->first_name);
    printf("\tLast Name: %s\n", student->last_name);
    printf("\tRoll Number: %d\n", student->roll_number);
    printf("\tGPA: %0.1f\n", student->GPA);
    printf("\tCourses IDs are\n");

    for(i = 0; i < COURSES_NUMBER; i++)
    {
        printf("\t\tCourse %d id: %d\n", i + 1, student->course_id[i]);
    }
}

static struct student_info *search_student_by_roll(FIFO_Buf_st *students_queue, int roll)
{
    int i;
    // we start from the tail because from base we can find that there is no data stored / removed
    Item *student = students_queue->tail;

    // Loop inside the queue
    for (i = 0; i < students_queue->counter; ++i)
    {
        // Check if we find the roll we search about to break
        if (student->roll_number == roll)

```

```

{
    // Get out form for loop
    break;
}

// Check if we reach the last item in the queue
if((student + 1) == (students_queue->base + students_queue->length))
{
    student = students_queue->base;
}
else
{
    // Just go to the next tail :D
    student++;
}
}

// Check if we couldn't find the roll number and we reach last item
if(i == students_queue->counter)
{
    // Return NULL
    student = NULL;
}

// Return the struct of the roll we found
return student;
}

FIFO_Status_st FIFO_enqueue(FIFO_Buf_st *fifo_buf, Item item)
{
    // Check parameters validity
    if(!fifo_buf || !fifo_buf->base || !fifo_buf->head || !fifo_buf->tail )
    {
        printf("FIFO Enqueue failed: Null is passed\n");

        return FIFO_NULL;
    }

    // Check if FIFO is Full ?!
    if((FIFO_is_full(fifo_buf)) == FIFO_FULL)
    {
        printf("FIFO Enqueue failed: FIFO is full\n");

        return FIFO_FULL;
    }

    // Enqueue new item
    *(fifo_buf->head) = item;

```

```

    if((fifo_buf->head + 1) == (fifo_buf->base + fifo_buf->length)) // Check if the head in the end of fifo to
start over
    {
        fifo_buf->head = fifo_buf->base;
    }
    else
    {
        fifo_buf->head++;
    }

    fifo_buf->counter++;

    return FIFO_NO_ERROR;
}

FIFO_Status_st FIFO_is_full(FIFO_Buf_st *fifo_buf)
{
    // Check parameters validity
    if(!fifo_buf || !fifo_buf->base || !fifo_buf->head || !fifo_buf->tail )
    {
        printf("FIFO is full failed: Null is passed\n");

        return FIFO_NULL;
    }

    // Check if the number of items matches the total buffer length
    if(fifo_buf->counter == fifo_buf->length)
    {
        return FIFO_FULL;
    }

    return FIFO_NOT_FULL;
}

FIFO_Status_st FIFO_is_empty(FIFO_Buf_st *fifo_buf)
{
    // Check parameters validity
    if(!fifo_buf || !fifo_buf->base || !fifo_buf->head || !fifo_buf->tail )
    {
        printf("FIFO is empty failed: Null is passed\n");

        return FIFO_NULL;
    }

    // Check if the number of items is zero
    if(fifo_buf->counter == 0)
    {
        return FIFO_EMPTY;
    }
}

```

```

    }
    return FIFO_NOT_EMPTY;
}

```

## Student\_Sys.h

```

#ifndef STUDENT_SYS_H_
#define STUDENT_SYS_H_
#include "stdio.h"
#include "string.h"
#define NAME_LENGTH 20
#define COURSES_NUMBER 5
// Student Structures
struct student_info {
    char first_name[NAME_LENGTH];
    char last_name[NAME_LENGTH];
    int roll_number;
    float GPA;
    int course_id[COURSES_NUMBER];
};
// Data type of buffer item
typedef struct student_info Item;
// Type Definition
typedef struct {
    Item *base;
    Item *head;
    Item *tail;
    int length;
    int counter;
}FIFO_Buf_st;
typedef enum {
    FIFO_NO_ERROR,
    FIFO_FULL,
    FIFO_NOT_FULL,
    FIFO_EMPTY,
    FIFO_NOT_EMPTY,
    FIFO_NULL
}FIFO_Status_st;
// Student Queue Initialization
FIFO_Status_st students_sys_init(FIFO_Buf_st *students_queue, Item *item, int length);
// Enter student data form file
void add_student_from_file(FIFO_Buf_st *students_queue);
// Enter student data form update file
void add_student_from_update_file(FIFO_Buf_st *students_queue);
// Enter student data manually from console
void add_student_manually(FIFO_Buf_st *students_queue);
// Get student date by its roll number
void find_student_by_roll(FIFO_Buf_st *students_queue);

```



```

// Get student date by its first name
void find_student_by_firstname(FIFO_Buf_st *students_queue);
// Get student date by its course
void find_student_by_course(FIFO_Buf_st *students_queue);
// Get students number in queue
void print_students_count(FIFO_Buf_st *students_queue);
// Delete student from the queue
void delete_student_by_roll(FIFO_Buf_st *students_queue);
// Update a specific data in queue
void update_student_by_roll(FIFO_Buf_st *students_queue);
// Update students and what we do in file
void update_student_file(FIFO_Buf_st *students_queue);
// Print all students in the queue
void show_students_info(FIFO_Buf_st *students_queue);
// Enqueue item in the FIFO given a data item
FIFO_Status_st FIFO_enqueue(FIFO_Buf_st *fifo_buf, Item item);
// Is the FIFO full ?!
FIFO_Status_st FIFO_is_full(FIFO_Buf_st *fifo_buf);
// Is the FIFO empty ?!
FIFO_Status_st FIFO_is_empty(FIFO_Buf_st *fifo_buf);
#endif /* STUDENT_SYS_H_ */

```

## Main.c

```

#include "Student_Sys.h"
#define STUDENTS_NUMBER 50
int main(void)
{
    int select_option;
    FIFO_Buf_st students_queue;
    struct student_info students_buffer[STUDENTS_NUMBER];
    // Solving I/O form console Problem
    setvbuf(stdout, NULL, _IONBF, 0);
    setvbuf(stderr, NULL, _IONBF, 0);
    students_sys_init(&students_queue, students_buffer, STUDENTS_NUMBER);
    printf("Welcome to the Student Management System\n");
    printf("\nDo you want to recover last database ?!\n");
    printf("\t1: Yes\n\t2: No\n");
    printf("Enter your option: ");
    scanf("%d",&select_option);

    switch (select_option)
    {
        case 1: add_student_from_update_file(&students_queue);break;
        case 2: break;
        default:break;
    }
    while(1)

```

```

{
    printf(" =====\n");
    printf("\n Choose on of the following options \n");
    printf("\n\t 1: Add Student Manually");
    printf("\n\t 2: Add Student From Text File");
    printf("\n\t 3: Find Student by Roll Number");
    printf("\n\t 4: Find Student by First Name");
    printf("\n\t 5: Find Student by Course ID");
    printf("\n\t 6: Print Students Number");
    printf("\n\t 7: Delete Student by Roll Number");
    printf("\n\t 8: Update Student by Roll Number");
    printf("\n\t 9: View Students");
    printf("\n\t 10: Exit");
    printf("\n\n Enter option number: ");
    scanf("%d",&select_option);
    printf(" ===== \n");
    switch(select_option)
    {
        case 1: add_student_manually(&students_queue);break;
        case 2: add_student_from_file(&students_queue);break;
        case 3: find_student_by_roll(&students_queue);break;
        case 4: find_student_by_firstname(&students_queue);break;
        case 5: find_student_by_course(&students_queue);break;
        case 6: print_students_count(&students_queue);break;
        case 7: delete_student_by_roll(&students_queue);break;
        case 8: update_student_by_roll(&students_queue);break;
        case 9: show_students_info(&students_queue);break;
        case 10: update_student_file(&students_queue); return 0;
        default: printf("\n Wrong Option: Try Again \n\n");break;
    }
}
return 0;
}

```

Students.txt

```

1 Mina Karam 2.8 1 2 3 4 5
1 Youstina Karam 3.3 5 21 73 92 36
3 Leila Mustafa 3.4 54 12 55 81 64
4 Koky Samy 3.6 5 21 55 92 64
5 Esraa Hegazy 3.6 54 12 55 81 64

```

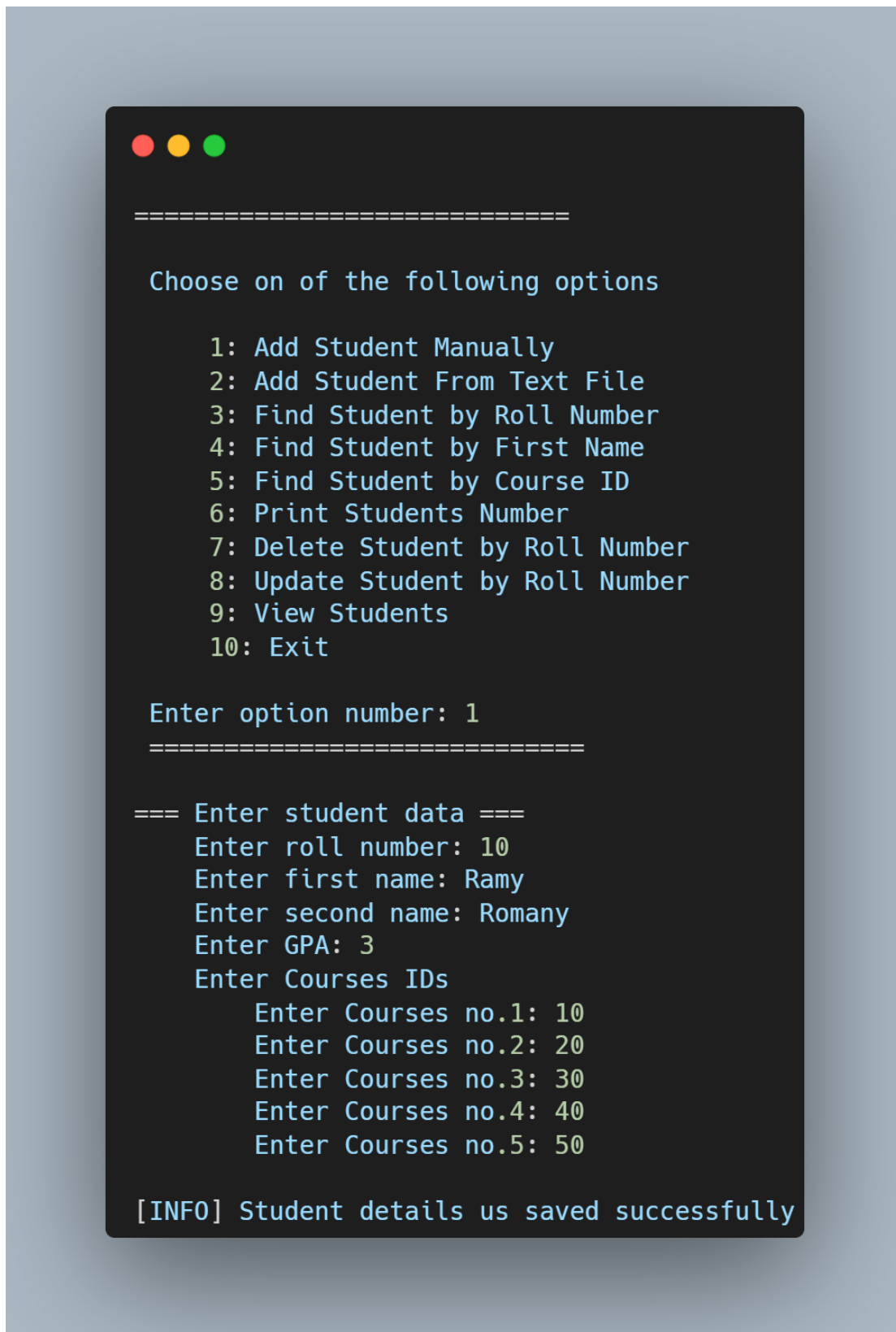
Updatestudent.txt

```

1 Mina Karam 2.8 1 2 3 4 5
3 Leila Mustafa 3.4 54 12 55 81 64
4 Koky Samy 3.6 5 21 55 92 64
5 Esraa Hegazy 3.6 54 12 55 81 64

```

# SCREENSHOTS





=====

Choose on of the following options

- 1: Add Student Manually
- 2: Add Student From Text File
- 3: Find Student by Roll Number
- 4: Find Student by First Name
- 5: Find Student by Course ID
- 6: Print Students Number
- 7: Delete Student by Roll Number
- 8: Update Student by Roll Number
- 9: View Students
- 10: Exit

Enter option number: 2

=====

[INFO] Roll number 1 is saved successfully

[ERROR] Roll number 1 is already taken

[INFO] Roll number 3 is saved successfully

[INFO] Roll number 4 is saved successfully

[INFO] Roll number 5 is saved successfully

[INFO] Students details are saved successfully

=====



=====

Choose on of the following options

- 1: Add Student Manually
- 2: Add Student From Text File
- 3: Find Student by Roll Number
- 4: Find Student by First Name
- 5: Find Student by Course ID
- 6: Print Students Number
- 7: Delete Student by Roll Number
- 8: Update Student by Roll Number
- 9: View Students
- 10: Exit

Enter option number: 3

=====

Enter roll number: 1

The student details are

First Name: Mina

Last Name: Karam

Roll Number: 1

GPA: 2.8

Courses IDs are

Course 1 id: 1

Course 2 id: 2

Course 3 id: 3

Course 4 id: 4

Course 5 id: 5

=====



=====

Choose on of the following options

- 1: Add Student Manually
- 2: Add Student From Text File
- 3: Find Student by Roll Number
- 4: Find Student by First Name
- 5: Find Student by Course ID
- 6: Print Students Number
- 7: Delete Student by Roll Number
- 8: Update Student by Roll Number
- 9: View Students
- 10: Exit

Enter option number: 4

=====

Enter First Name: Mina

The student details are

First Name: Mina

Last Name: Karam

Roll Number: 1

GPA: 2.8

Courses IDs are

Course 1 id: 1

Course 2 id: 2

Course 3 id: 3

Course 4 id: 4

Course 5 id: 5

=====



=====

Choose on of the following options

- 1: Add Student Manually
- 2: Add Student From Text File
- 3: Find Student by Roll Number
- 4: Find Student by First Name
- 5: Find Student by Course ID
- 6: Print Students Number
- 7: Delete Student by Roll Number
- 8: Update Student by Roll Number
- 9: View Students
- 10: Exit

Enter option number: 5

=====

Enter Course ID: 55

The student details are

First Name: Leila  
Last Name: Mustafa  
Roll Number: 3  
GPA: 3.4

Courses IDs are

Course 1 id: 54  
Course 2 id: 12  
Course 3 id: 55  
Course 4 id: 81  
Course 5 id: 64

The student details are

First Name: Koky  
Last Name: Samy  
Roll Number: 4  
GPA: 3.6

Courses IDs are

Course 1 id: 5  
Course 2 id: 21  
Course 3 id: 55  
Course 4 id: 92  
Course 5 id: 64

The student details are

First Name: Esraa  
Last Name: Hegazy  
Roll Number: 5  
GPA: 3.6

Courses IDs are

Course 1 id: 54  
Course 2 id: 12  
Course 3 id: 55  
Course 4 id: 81  
Course 5 id: 64

[INFO] Total number of students enrolled: 3

=====



=====

Choose on of the following options

- 1: Add Student Manually
- 2: Add Student From Text File
- 3: Find Student by Roll Number
- 4: Find Student by First Name
- 5: Find Student by Course ID
- 6: Print Students Number
- 7: Delete Student by Roll Number
- 8: Update Student by Roll Number
- 9: View Students
- 10: Exit

Enter option number: 6

=====

[INFO] Total number of students is 5  
[INFO] You can add up to 50 students  
[INFO] You can add 45 more students

=====





=====

Choose on of the following options

- 1: Add Student Manually
- 2: Add Student From Text File
- 3: Find Student by Roll Number
- 4: Find Student by First Name
- 5: Find Student by Course ID
- 6: Print Students Number
- 7: Delete Student by Roll Number
- 8: Update Student by Roll Number
- 9: View Students
- 10: Exit

Enter option number: 7

=====

Enter roll number: 10

[INFO] The Roll Number is removed successfully

=====

Enter roll number: 20

[ERROR] This Roll Number 20 not found

=====



=====

Choose on of the following options

- 1: Add Student Manually
- 2: Add Student From Text File
- 3: Find Student by Roll Number
- 4: Find Student by First Name
- 5: Find Student by Course ID
- 6: Print Students Number
- 7: Delete Student by Roll Number
- 8: Update Student by Roll Number
- 9: View Students
- 10: Exit

Enter option number: 8

=====

Enter roll number: 1

==== Student data ====

The student details are

First Name: Mina

Last Name: Karam

Roll Number: 1

GPA: 2.8

Courses IDs are

Course 1 id: 1

Course 2 id: 2

Course 3 id: 3

Course 4 id: 4

Course 5 id: 5

Which date do you want to change ?

1: The Roll Number

2: The First Name

3: The Second Name

4: The GPA Score

5: The Courses ID

Enter your option: 1

Enter the new roll number: 10

[INFO] The Roll Number 10 in updated successfully

=====

Enter your option: 2

Enter the new first name: Youstina

[INFO] The First Name Youstina in updated successfully

=====

Enter your option: 3

Enter the new last name: Mina

[INFO] The Last Name Mina in updated successfully

=====

Enter your option: 4

Enter the new GPA: 4

[INFO] The GPA Score 4.0 in updated successfully

=====

Enter your option: 5

Enter the course number from 1 to 5: 3

Enter the new course id: 20

[INFO] The Course ID 20 in updated successfully



=====

Choose on of the following options

- 1: Add Student Manually
- 2: Add Student From Text File
- 3: Find Student by Roll Number
- 4: Find Student by First Name
- 5: Find Student by Course ID
- 6: Print Students Number
- 7: Delete Student by Roll Number
- 8: Update Student by Roll Number
- 9: View Students
- 10: Exit

Enter option number: 9

=====

All Students

The student details are

First Name: Leila

Last Name: Mustafa

Roll Number: 3

GPA: 3.4

Courses IDs are

Course 1 id: 54

Course 2 id: 12

Course 3 id: 55

Course 4 id: 81

Course 5 id: 64

The student details are

First Name: Koky

Last Name: Samy

Roll Number: 4

GPA: 3.6

Courses IDs are

Course 1 id: 5

Course 2 id: 21

Course 3 id: 55

Course 4 id: 92

Course 5 id: 64

The student details are

First Name: Esraa

Last Name: Hegazy

Roll Number: 5

GPA: 3.6

Courses IDs are

Course 1 id: 54

Course 2 id: 12

Course 3 id: 55

Course 4 id: 81

Course 5 id: 64

The student details are

First Name: Youstina

Last Name: Mina

Roll Number: 10

GPA: 4.0

Courses IDs are

Course 1 id: 1

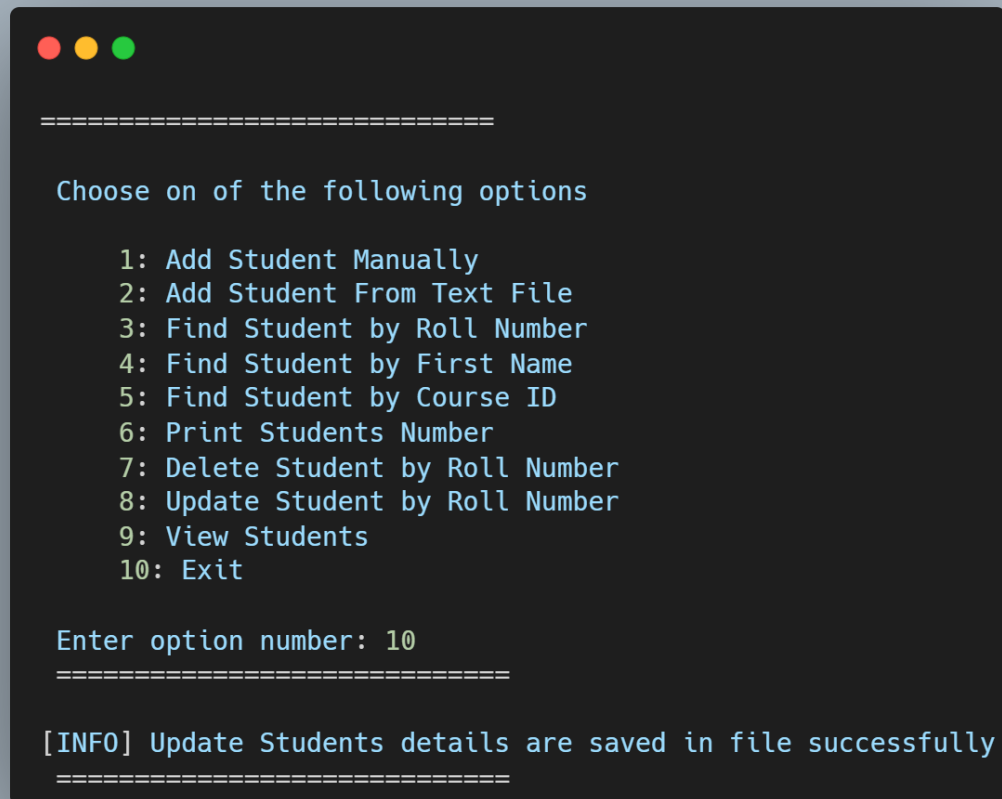
Course 2 id: 2

Course 3 id: 20

Course 4 id: 4

Course 5 id: 5

=====



```
=====

Choose on of the following options

1: Add Student Manually
2: Add Student From Text File
3: Find Student by Roll Number
4: Find Student by First Name
5: Find Student by Course ID
6: Print Students Number
7: Delete Student by Roll Number
8: Update Student by Roll Number
9: View Students
10: Exit

Enter option number: 10
=====

[INFO] Update Students details are saved in file successfully
=====
```

# CONCLUSION

In summary, the `Student_sys.h` header file serves as the backbone of our student information management system, encapsulating essential elements such as macros, data structures, and function prototypes. Through meticulous structuring, this header file ensures the program's organization, readability, and ease of use. The struct `student_info` provides a standardized format for storing student data, and the use of macros and type definitions enhances code clarity and simplifies complex operations.

## **Future Scope:**

Looking ahead, the student information management system can be further enhanced and expanded to cater to evolving educational needs. Future developments may include the integration of advanced algorithms for data analysis, implementation of user authentication and authorization mechanisms to enhance security, incorporation of data visualization tools for performance analysis, and the adaptation of cloud-based solutions for seamless accessibility and scalability. Additionally, the system can be extended to support multimedia data, real-time notifications, and predictive analytics, providing a holistic platform for educators, administrators, and students.

Moreover, the system's user interface can be refined for intuitive user experiences, and mobile applications can be developed to ensure accessibility on various devices. Collaboration features, such as discussion forums and collaborative project spaces, can be integrated to foster a dynamic learning environment. Embracing emerging technologies such as artificial intelligence and machine learning can enable the system to provide personalized learning paths and intelligent recommendations, enhancing the overall educational journey.

In essence, the future scope of the student information management system is vast and dynamic, promising continuous innovation and improvements to meet the ever-changing demands of modern education. Through ongoing research and development, the system can evolve into a sophisticated educational ecosystem, empowering educational institutions and stakeholders to nurture the next generation of learners effectively.