Mastering Embedded System Online Diploma

www.learn-in-depth.com

First Term (Final Project 2)

Student Management System Report

Eng. Mohamed Hazem Yahya Mahrous Ali

Content: -

Introduction

The Student Management System is a C programming project designed to manage student information using a First-In-First-Out (FIFO) queue data structure. This system allows users to perform various tasks related to student data, such as adding students manually or from a saved data file, searching for students by different criteria, updating student information, deleting students, and viewing the list of all students.

Project Structure

The project consists of three main source code files: **main.c**, **buffer.h**, and **buffer.c**. Below is an overview of each file's purpose and functionality:

main.c

This is the main program file that drives the Student Management System. It contains the **main** function, which provides a user interface for interacting with the system. Users can choose from a menu of tasks to perform, such as adding students, searching for students, updating student data, and more.

Key functionalities in **main.c**:

- User interface and menu for task selection.
- Integration with the **buffer.h** functions to manipulate the student data.

buffer.h

This header file defines the data structures, function prototypes, and status codes used throughout the project.

buffer.c

This source code file contains the implementations of the functions declared in **buffer.h**. These functions provide the core logic for managing student data within the FIFO queue. They handle tasks such as adding, searching, updating, and deleting students, as well as printing student information.

FIFO Queue

The project implements a FIFO queue using the **FIFO_Queue_st** structure, which is defined in **buffer.h**. The queue has the following properties:

- **length**: Total length of the queue.
- **count**: Number of elements in the queue.
- **head**: Pointer to the first element (head) of the queue.
- **tail**: Pointer to the last element (tail) of the queue.
- **base**: Pointer to the base of the queue.

Functions

The project includes various functions to perform operations on the FIFO queue, including:

- Initializing the queue
- Enqueuing students
- Checking if the queue is full or empty
- Adding students manually or from a predefined dataset
- Getting students by ID, first name, or course ID
- Deleting students by ID
- Updating student information
- Getting the length of the queue
- Printing the list of students and individual student information

Code Implementation

1-Main.c

```c
#include "buffer.h"

FIFO_Queue_st QUEUE_UART, QUEUE_TEST;
struct Sstudent UART_BUF[QUEUE_LEN];

int main(void) {
    struct Sstudent UART_BUF_TEST[QUEUE_LEN];
    QUEUE_INIT(&QUEUE_TEST, UART_BUF_TEST, QUEUE_LEN);

    int choise, while_loop = 1;

    printf("Welcome to the Student Management System");
    while(while_loop){
        printf("\n\nChoose The Task You Want to Perform\n");
        printf("1: Add Student Details Manually\n");
        printf("2: Add Student Details From the Saved Data\n");
        printf("3: Find Student Details By ID\n");
        printf("4: Find Student Details By First Name\n");
        printf("5: Find Student Details By Course ID\n");
        printf("6: Find the Total Number of Students\n");
        printf("7: Delete Students Details Using The ID\n");
        printf("8: Update Students Details Using The ID\n");
        printf("9: Show All Informations\n");
        printf("10: To Exit\n");
        printf("\nEnter Your Choise to perform the task: ");
        fflush(stdout);
        scanf("%d", &choise);
        fflush(stdin);

        switch(choise){
        case 1:
            ADD_STUDENT_MANUALY(&QUEUE_TEST);
            break;
        case 2:
            ADD_STUDENT_FROM_FILE(&QUEUE_TEST);
            break;
        case 3:
            GET_STUDENT_BY_ROLL(&QUEUE_TEST);
            break;
        case 4:
            GET_STUDENT_BY_FNAME(&QUEUE_TEST);
            break;
        case 5:
            GET_STUDENT_BY_COURSE(&QUEUE_TEST);
            break;
        case 6:
            GET_LENGTH(&QUEUE_TEST);
            break;
        case 7:
            DELETE_STUDENT(&QUEUE_TEST);
            break;
        case 8:
            UPDATE_STUDENT(&QUEUE_TEST);
            break;
        case 9:
            PRINT_LIST(&QUEUE_TEST);
            break;
        case 10:
            while_loop = 0;
            break;
        default:
            printf("\nWrong Option");
            break;
        }
    }
    return EXIT_SUCCESS;
}
```

2-Buffer.h

```c
#ifndef BUFFER_H_
#define BUFFER_H_

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>

// User Configuration ==================
#define Course_Num 5        // Defines the number of courses for every student
#define student_num 10       // Defines how many students can be added to the saved file
#define QUEUE_LEN 50        // Defines the length of the data structure used for the queue


// Definition of a structure to hold student data
struct Saved_Date {
    char fname[student_num][50];    // First names of students
    char lname[student_num][50];    // Last names of students
    int ID[student_num];            // Student IDs
    float GPA[student_num];         // GPAs of students
    int CourseID[student_num][Course_Num];    // IDs of courses taken by students
};

// Extern declaration of the structure to store saved data
extern struct Saved_Date DATA_FILE;

// Definition of a structure to represent a student
struct Sstudent {
    char fname[50];         // First name of student
    char lname[50];         // Last name of student
    int ID;                 // Student ID
    float GPA;              // Student GPA
    int CourseID[Course_Num];    // IDs of courses taken by student
};
// Definition of a FIFO (First-In-First-Out) queue structure
typedef struct {
    unsigned int length;        // Total length of the queue
    unsigned int count;         // Number of elements in the queue
    struct Sstudent* head;      // Pointer to the first element (head) of the queue
    struct Sstudent* tail;      // Pointer to the last element (tail) of the queue
    struct Sstudent* base;      // Pointer to the base of the queue
} FIFO_Queue_st;
```

```c
// Enum to define different status codes for the FIFO queue operations
typedef enum {
    FIFO_ERROR,
    FIFO_NO_ERROR,
    FIFO_FULL,
    FIFO_NOT_FULL,
    FIFO_EMPTY,
    FIFO_NOT_EMPTY,
    FIFO_NULL
} FIFO_STATUS_EN;



// Function prototypes for FIFO queue operations
FIFO_STATUS_EN QUEUE_INIT(FIFO_Queue_st *QUEUE, struct Sstudent *student, int length);
FIFO_STATUS_EN ENQUEUE(FIFO_Queue_st *QUEUE, struct Sstudent student);

FIFO_STATUS_EN IS_QUEUE_FULL(FIFO_Queue_st *QUEUE);
FIFO_STATUS_EN IS_QUEUE_EMPTY(FIFO_Queue_st *QUEUE);

FIFO_STATUS_EN ADD_STUDENT_FROM_FILE(FIFO_Queue_st *QUEUE);
FIFO_STATUS_EN ADD_STUDENT_MANUALY(FIFO_Queue_st *QUEUE);

FIFO_STATUS_EN GET_STUDENT_BY_ROLL(FIFO_Queue_st *QUEUE);
FIFO_STATUS_EN GET_STUDENT_BY_FNAME(FIFO_Queue_st *QUEUE);
FIFO_STATUS_EN GET_STUDENT_BY_COURSE(FIFO_Queue_st *QUEUE);

FIFO_STATUS_EN DELETE_STUDENT(FIFO_Queue_st *QUEUE);
FIFO_STATUS_EN UPDATE_STUDENT(FIFO_Queue_st *QUEUE);

FIFO_STATUS_EN GET_LENGTH(FIFO_Queue_st *QUEUE);
FIFO_STATUS_EN PRINT_LIST(FIFO_Queue_st *QUEUE);
FIFO_STATUS_EN PRINT_STUDENT(FIFO_Queue_st *QUEUE, int Stud_ID);

#endif /* BUFFER_H_ */
```

3-Buffer.c

```c
#include "buffer.h"

// write the saved data to add automatically
struct Saved_Date DATA_FILE = {
        .fname = {
                "Mohamed", "Ahmed", "Mahmoud", "Layla", "Sara",
                "Fatima", "Omar", "Nadia", "Khalid", "Aisha"
        },
        .lname = {
                "Hazem", "Mahmoud", "Attia", "Smith", "Johnson",
                "Ali", "Abdullah", "Khan", "Ahmed", "Yusuf"
        },
        .ID = {1, 1, 3, 4, 5, 6, 7, 8, 9, 10},
        .GPA = {3.2, 2.5, 4, 3.8, 3.9, 3.0, 3.7, 3.4, 3.1, 3.6},
        .CourseID = {
                {572, 319, 856, 437, 648},   // Courses for Mohamed
                {319, 856, 437, 648, 182},   // Courses for Ahmed
                {856, 437, 648, 182, 725},   // Courses for Mahmoud
                {437, 648, 182, 725, 921},   // Courses for Layla
                {648, 182, 725, 921, 563},   // Courses for Sara
                {182, 725, 921, 563, 104},   // Courses for Fatima
                {725, 921, 563, 104, 572},   // Courses for Omar
                {921, 563, 104, 572, 319},   // Courses for Nadia
                {563, 104, 572, 319, 856},   // Courses for Khalid
                {104, 572, 319, 856, 437}    // Courses for Aisha
        }
};
```

```c
// This function initializes a FIFO queue structure with the provided arguments.
// It ensures that the provided pointers are valid and the length is non-zero.
// If successful, it initializes the queue properties and returns a success status.
// If any of the arguments is invalid, it prints an error message and returns an appropriate error status
FIFO_STATUS_EN QUEUE_INIT(FIFO_Queue_st *QUEUE, struct Sstudent *student, int length) {
    if (QUEUE == NULL || student == NULL || length == 0) {
        printf("\n[ERROR] Queue Init Failed, Make Sure That The Arguments Are Right \n");
        return FIFO_NULL;
    }
    QUEUE->base = student;
    QUEUE->head = QUEUE->base;
    QUEUE->tail = QUEUE->base;
    QUEUE->length = length;
    QUEUE->count = 0;
    printf("[INFO] Queue Init is Done \n\n");
    return FIFO_NO_ERROR;
}



// This function checks whether the provided FIFO queue is full or not.
// It verifies if the queue's base, head, and tail pointers are valid.
// If the queue is full, it prints a message and returns the appropriate status.
// If the queue is not full, it prints a message and returns the appropriate status.
FIFO_STATUS_EN IS_QUEUE_FULL(FIFO_Queue_st* QUEUE) {
    if (QUEUE->base == NULL || QUEUE->head == NULL || QUEUE->tail == NULL) {
        printf("\n[ERROR] Queue Doesn't Exist, Please Enter Valid FIFO\n\n");
        return FIFO_NULL;
    }
    if (QUEUE->count == QUEUE->length) {
        printf("[INFO] Queue Is Full\n");
        return FIFO_FULL;
    } else {
        printf("[INFO] Queue Is Not Full\n");
        return FIFO_NOT_FULL;
    }
}
```

```c
// This function checks whether the provided FIFO queue is empty or not.
// It verifies if the queue's base, head, and tail pointers are valid.
// If the queue is empty, it prints a message and returns the appropriate status.
// If the queue is not empty, it prints a message and returns the appropriate status.
FIFO_STATUS_EN IS_QUEUE_EMPTY(FIFO_Queue_st *QUEUE) {
    if (QUEUE->base == NULL || QUEUE->head == NULL || QUEUE->tail == NULL) {
        printf("\n[ERROR] Queue Doesn't Exist, Please Enter Valid FIFO\n\n");
        return FIFO_NULL;
    }
    if (QUEUE->count == 0) {
        printf("[INFO] Queue Is Empty\n");
        return FIFO_FULL; // This should be changed to FIFO_EMPTY
    } else {
        printf("[INFO] Queue Is Not Empty\n");
        return FIFO_NOT_FULL; // This should be changed to FIFO_NOT_EMPTY
    }
}
```

```c
// Function to enqueue a student into the FIFO queue
FIFO_STATUS_EN ENQUEUE(FIFO_Queue_st *QUEUE, struct Sstudent student) {
    // Check if the queue or its components are not properly initialized
    if (QUEUE->base == NULL || QUEUE->head == NULL || QUEUE->tail == NULL) {
        printf("\n[ERROR] Queue Doesn't Exist, Please Enter Valid FIFO\n\n");
        return FIFO_NULL; // Return an error code indicating NULL queue
    }

    // Check if the queue is full (head has reached the end of the allocated memory)
    if (QUEUE->head == QUEUE->base + QUEUE->length) {
        printf("\n[ERROR] Queue is Full\n");
        return FIFO_FULL; // Return an error code indicating full queue
    }

    // Declare a pointer to a struct Sstudent named temp and initialize it with the tail pointer of the queue.
    struct Sstudent* temp = QUEUE->tail;

    // Loop through the queue to check if the entered student ID is already taken.
    for(int i = 0; i<QUEUE->count; i++){
        // Compare the entered ID with the ID of the student pointed to by temp.
        if(student.ID == temp->ID){
            // If the ID is already taken, print an error message and return FIFO_ERROR.
            printf("\n[ERROR] Roll number %d is already taken\n", student.ID);
            return FIFO_ERROR;
        }
        temp++; // Move the temp pointer to the next student in the queue.
    }
    // Assign the student structure to the current location pointed to by head
    *(QUEUE->head) = student;

    // Move the head pointer to the next location
    QUEUE->head++;

    // Increment the count of elements in the queue
    QUEUE->count++;

    // Return a success code indicating successful enqueue
    return FIFO_NO_ERROR;
}
```

```c
FIFO_STATUS_EN ADD_STUDENT_MANUALY(FIFO_Queue_st *QUEUE){
    // Check if the base, head, or tail pointers of the given queue are NULL.
    if (QUEUE->base == NULL || QUEUE->head == NULL || QUEUE->tail == NULL) {
        // If any of the pointers is NULL, print an error message and return FIFO_NULL.
        printf("\n[ERROR] Queue Doesn't Exist, Please Enter Valid FIFO\n\n");
        return FIFO_NULL;
    }

    // Declare a struct Sstudent variable named NewStudent to store student information.
    struct Sstudent NewStudent;
    // Declare an integer variable i for later use.

    // Print a prompt to enter student data.
    printf("\n=== Enter student data ===\n");
    // Prompt the user to enter the student's ID and read it from input.
    printf("Enter ID: ");
    fflush(stdout); // Flush the output buffer to ensure the prompt is displayed.
    scanf("%d", &NewStudent.ID); // Read the entered value into NewStudent.ID.
    fflush(stdin); // Flush the input buffer to clear any remaining newline characters.

    // Declare a pointer to a struct Sstudent named temp and initialize it with the tail pointer of the queue.
    struct Sstudent* temp = QUEUE->tail;

    // Loop through the queue to check if the entered student ID is already taken.
    for(int i = 0; i<QUEUE->count; i++){
        // Compare the entered ID with the ID of the student pointed to by temp.
        if(NewStudent.ID == temp->ID){
            // If the ID is already taken, print an error message and return FIFO_ERROR.
            printf("\n[ERROR] Roll number %d is already taken\n", NewStudent.ID);
            return FIFO_ERROR;
        }
        temp++; // Move the temp pointer to the next student in the queue.
    }

    // Prompt the user to enter the student's first name and read it from input.
    printf("Enter First Name: ");
    fflush(stdout);
    scanf("%s", NewStudent.fname);
    fflush(stdin);
```

```c
    // Prompt the user to enter the student's first name and read it from input.
    printf("Enter First Name: ");
    fflush(stdout);
    scanf("%s", NewStudent.fname);
    fflush(stdin);

    // Prompt the user to enter the student's last name and read it from input.
    printf("Enter Second Name: ");
    fflush(stdout);
    scanf("%s", NewStudent.lname);
    fflush(stdin);

    // Prompt the user to enter the student's GPA and read it from input.
    printf("Enter GPA: ");
    fflush(stdout);
    scanf("%f", &NewStudent.GPA);
    fflush(stdin);

    // Prompt the user to enter course IDs for the student.
    printf("Enter Courses IDs\n");
    // Loop through the course IDs and prompt the user to enter each one.
    for (int i = 0; i < Course_Num; ++i)
    {
        printf("\tEnter Courses no.%d: ",i+1);
        fflush(stdout);
        scanf("%d", &NewStudent.CourseID[i]);
        fflush(stdin);
    }

    // Attempt to enqueue the NewStudent into the queue using the ENQUEUE function.
    if(ENQUEUE(QUEUE, NewStudent) == FIFO_NO_ERROR)
    {
        // If enqueueing is successful, print an info message and return FIFO_NO_ERROR.
        printf("\n[INFO] Enqueueing Student %s %s Details in Queue is Done\n", NewStudent.fname , NewStudent.lname);
        return FIFO_NO_ERROR;
    }
    else
    {
        // If enqueueing fails, print an error message and return FIFO_ERROR.
        printf("\n[ERROR] Enqueueing Student %s %s Failed\n", NewStudent.fname , NewStudent.lname);
        return FIFO_ERROR;
    }
```

```c
FIFO_STATUS_EN DELETE_STUDENT(FIFO_Queue_st *QUEUE) {

    // Check if the queue pointers are valid
    if (QUEUE->base == NULL || QUEUE->head == NULL || QUEUE->tail == NULL) {
        printf("\n[ERROR] Queue Doesn't Exist, Please Enter a Valid FIFO\n\n");
        return FIFO_NULL;
    }

    // Create a temporary pointer to navigate the queue
    struct Sstudent* Temp = QUEUE->tail;

    // Declare a variable to store the ID of the student to be deleted
    int DelStudent;

    // Prompt the user for the student ID to delete
    printf("\n[INFO] Enter ID to delete: ");
    fflush(stdout); // Flush output buffer to display the prompt
    scanf("%d", &DelStudent); // Read the ID from user input
    fflush(stdin); // Flush input buffer to clear newline character

    // Loop through the queue to find the student with the specified ID
    for (int i = 0; i < QUEUE->count; i++) {
        // Check if the current student's ID matches the one to delete
        if (DelStudent == Temp->ID) {
            // Print information about the student being deleted
            printf("\n[INFO] Student with ID %d which is %s %s has been deleted\n", DelStudent, Temp->fname, Temp->lname);

            // If found, shift all subsequent students one position back
            for (int j = i; j < QUEUE->count - 1; j++) {
                *Temp = *(Temp + 1); // Shift the data of the next student to the current position
                Temp++; // Move the temporary pointer to the next position
            }
            QUEUE->count--; // Decrement the queue count to indicate a deleted student
            return FIFO_NO_ERROR; // Return success status
        }
        Temp++; // Move the temporary pointer to the next student in the queue
    }

    // If the student with the specified ID was not found in the queue
    printf("\n[ERROR] Student with ID %d not found in the queue\n", DelStudent);
    return FIFO_ERROR; // Return error status
}
```

```c
// Function to add students from a file to a FIFO queue
FIFO_STATUS_EN ADD_STUDENT_FROM_FILE(FIFO_Queue_st *QUEUE) {
    // Checking if the queue structure pointers are valid
    if (QUEUE->base == NULL || QUEUE->head == NULL || QUEUE->tail == NULL) {
        printf("\n[ERROR] Queue Doesn't Exist, Please Enter Valid FIFO\n\n");
        return FIFO_NULL;
    }

    // Loop through each student in the file
    for (int i = 0; i < student_num; i++) {
        // Creating a new student structure
        struct Sstudent NewStudent;

        // Copying first name, last name, ID, and GPA from the data file to the new student
        strcpy(NewStudent.fname, DATA_FILE.fname[i]);
        strcpy(NewStudent.lname, DATA_FILE.lname[i]);
        NewStudent.ID = DATA_FILE.ID[i];
        NewStudent.GPA = DATA_FILE.GPA[i];

        // Loop through each course for the current student
        for (int j = 0; j < Course_Num; j++) {
            // Copying course ID from data file to the new student's course ID array
            NewStudent.CourseID[j] = DATA_FILE.CourseID[i][j];
        }

        // Attempt to enqueue the new student into the queue
        if (ENQUEUE(QUEUE, NewStudent) == FIFO_NO_ERROR) {
            printf("\n[INFO] Enqueueing Student %s %s Details in Queue is Done\n", NewStudent.fname, NewStudent.lname);
        } else {
            printf("\n[ERROR] Enqueueing Student %s %s Failed\n", NewStudent.fname, NewStudent.lname);
        }
    }
    // Return success status
    return FIFO_NO_ERROR;
}
```

```c
FIFO_STATUS_EN UPDATE_STUDENT(FIFO_Queue_st *QUEUE){
    // Function to update student information in a FIFO queue.
    if (QUEUE->base == NULL || QUEUE->head == NULL || QUEUE->tail == NULL) {
        // Check if the queue pointers are valid.
        printf("\n[ERROR] Queue Doesn't Exist, Please Enter a Valid FIFO\n\n");
        return FIFO_NULL; // Return error status if queue doesn't exist.
    }

    struct Sstudent* Temp = QUEUE->tail; // Create a temporary pointer to traverse the queue.
    int UpdStudent, choise, course_choise, id_found = 0; // Declare variables to store user inputs.

    printf("\n[INFO] Enter ID to Update: ");
    fflush(stdout); // Flush the output buffer to display the prompt.
    scanf("%d", &UpdStudent); // Read the ID of the student to be updated.
    fflush(stdin); // Flush the input buffer to clear any remaining characters.


    for (int i = 0; i < QUEUE->count; i++) {
        // Loop to find the student with the specified ID in the queue.
        if (UpdStudent == Temp->ID) {
            id_found = 1;
            break; // Exit the loop when the desired student is found.
        } else {
            Temp++; // Move the temporary pointer to the next student in the queue.
        }
    }
    // Check if the 'id_found' flag is still 0, indicating that the ID was not found in the queue.
    if(!id_found){
        // Print an error message indicating that the ID could not be found and return FIFO_ERROR error code.
        printf("\n[ERROR] This ID Can't Be Found ");
        return FIFO_ERROR;
    }


    printf("\nChoose Which Attribute you want to change\n");
    // Display options for attributes to update.
    printf("1: First Name\n");
    printf("2: Last Name\n");
    printf("3: ID\n");
    printf("4: GPA\n");
    printf("5: Course\n");
    printf("Enter Your Choise : ");
    fflush(stdout);
    scanf("%d", &choise); // Read user's choice of attribute to update.
    fflush(stdin);
```

```c
    switch (choise) {
    case 1:
        // Update the student's first name.
        printf("\nEnter The New First Name : ");
        fflush(stdout);
        scanf("%s", Temp->fname);
        fflush(stdin);
        break;
    case 2:
        // Update the student's last name.
        printf("\nEnter The New Last Name : ");
        fflush(stdout);
        scanf("%s", Temp->lname);
        fflush(stdin);
        break;
    case 3:
        // Update the student's ID.
        printf("\nEnter The New ID : ");
        fflush(stdout);
        scanf("%d", &Temp->ID);
        fflush(stdin);
        break;
    case 4:
        // Update the student's GPA.
        printf("\nEnter The New GPA : ");
        fflush(stdout);
        scanf("%f", &Temp->GPA);
        fflush(stdin);
        break;
    case 5:
        // Update the student's course ID.
        printf("\nChoose Which Course you want to change\n");
        // Display options for available courses.
        for (int i = 0; i < Course_Num; i++) {
            printf("%d: Course Number %d\n", i + 1, i + 1);
        }
        printf("Enter Your Choise : ");
        fflush(stdout);
        scanf("%d", &course_choise); // Read user's choice of course to update.
        fflush(stdin);

        printf("Enter The New Value : ");
        fflush(stdout);
        scanf("%d", &Temp->CourseID[course_choise - 1]); // Update the selected course ID.
        fflush(stdin);
```

```c
// Function definition: GET_STUDENT_BY_ROLL takes a pointer to a FIFO_Queue_st struct as an argument.
FIFO_STATUS_EN GET_STUDENT_BY_ROLL(FIFO_Queue_st *QUEUE){
    // Check if the base, head, or tail pointers of the queue are NULL.
    if (QUEUE->base == NULL || QUEUE->head == NULL || QUEUE->tail == NULL) {
        // If any of the pointers are NULL, print an error message and return FIFO_NULL error code.
        printf("\n[ERROR] Queue Doesn't Exist, Please Enter Valid FIFO\n\n");
        return FIFO_NULL;
    }

    // Declare a pointer to a struct Sstudent and initialize it with the value of QUEUE->tail.
    struct Sstudent* Temp;
    Temp = QUEUE->tail;

    // Declare variables for ID search and 'found' flag to track whether the ID was found in the queue.
    int ID_Search, found = 0;

    // Prompt the user to enter the ID to search for.
    printf("\n[INFO] Enter ID to Search for: ");
    fflush(stdout);
    // Read the ID from user input.
    scanf("%d", &ID_Search);
    fflush(stdin);

    // Loop through the queue to search for the specified ID.
    for(int i = 0; i < QUEUE->count; i++){
        // Check if the current ID matches the search ID.
        if(ID_Search == Temp->ID){
            // If a match is found, print the student's information using PRINT_STUDENT function.
            PRINT_STUDENT(QUEUE, Temp->ID);
            // Set 'found' flag to 1 to indicate a match was found.
            found = 1;
        }
        // Move the Temp pointer to the next element in the queue.
        Temp++;
    }

    // Check if the 'found' flag is still 0, indicating that the ID was not found in the queue.
    if(!found){
        // Print an error message indicating that the ID could not be found and return FIFO_ERROR error code.
        printf("\n[ERROR] This ID Can't Be Found ");
        return FIFO_ERROR;
    }

    // If the ID was found, return FIFO_NO_ERROR to indicate successful completion.
    return FIFO_NO_ERROR;
}
```

```c
// Function to retrieve a student by first name from a FIFO queue

FIFO_STATUS_EN GET_STUDENT_BY_FNAME(FIFO_Queue_st *QUEUE) {
    // Check if the queue's base, head, or tail pointers are NULL
    if (QUEUE->base == NULL || QUEUE->head == NULL || QUEUE->tail == NULL) {
        printf("\n[ERROR] Queue Doesn't Exist, Please Enter Valid FIFO\n\n");
        return FIFO_NULL; // Return an error code for null queue
    }

    struct Sstudent* Temp; // Declare a pointer to a student structure
    Temp = QUEUE->tail; // Initialize Temp with the tail pointer of the queue

    char Name_Search[50]; // Declare an array to store the search name

    printf("\n[INFO] Enter Name to Search for: "); // Prompt user to enter a name
    fflush(stdout); // Flush the output buffer to ensure the prompt is displayed
    scanf("%s", Name_Search); // Read the search name from user input
    fflush(stdin); // Flush the input buffer to clear any remaining characters

    for (int i = 0; i < QUEUE->count; i++) {
        // Compare the search name with the first name of the student pointed to by Temp
        if (stricmp(Name_Search, Temp->fname) == 0) {
            PRINT_STUDENT(QUEUE, Temp->ID); // Print student information using ID
        } else {
            printf("\n[ERROR] This Name Can't Be Found "); // Indicate that the name was not found
            return FIFO_ERROR; // Return an error code for name not found
        }
        Temp++; // Move Temp to the next student in the queue
    }
    return FIFO_NO_ERROR; // Return success code, indicating the search completed
}
```

```c
FIFO_STATUS_EN GET_STUDENT_BY_COURSE(FIFO_Queue_st *QUEUE){
    // Function to search for students by a given course ID in a FIFO queue.
    if (QUEUE->base == NULL || QUEUE->head == NULL || QUEUE->tail == NULL) {
        // Check if the queue's base, head, or tail pointers are NULL.
        // If any of them is NULL, it indicates that the queue is not properly initialized.
        printf("\n[ERROR] Queue Doesn't Exist, Please Enter Valid FIFO\n\n");
        return FIFO_NULL;  // Return an error code for a null queue.
    }

    struct Sstudent* Temp;  // Declare a pointer to a structure of type Sstudent.
    Temp = QUEUE->tail;  // Initialize Temp pointer with the tail pointer of the queue.
    int j;  // Declare an integer variable j.

    int Course_Search, course_count = 0;  // Declare variables for storing course ID to search and a counter.

    printf("\n[INFO] Enter Course ID to Search for: ");  // Prompt the user to enter a course ID.
    fflush(stdout);  // Flush the output buffer to display the prompt.
    scanf("%d", &Course_Search);  // Read the entered course ID from the user.
    fflush(stdin);  // Flush the input buffer.

    for(int i = 0; i < QUEUE->count; i++){
        // Loop through each student in the queue (number of times specified by queue count).
        for(j = 0; j < Course_Num; j++){
            // Loop through each course in the student's course list (Course_Num times).
            if(Course_Search == Temp->CourseID[j]){
                // Check if the entered course ID matches any course in the current student's course list.
                printf("\n[INFO] The Student %s %s which ID is %d Has The Course ID %d",
                        Temp->fname, Temp->lname, Temp->ID, Temp->CourseID[j]);
                course_count++;  // Increment the course_count since a match was found.
            }
        }
        Temp++;  // Move the Temp pointer to the next student in the queue.
    }
    if(course_count == 0){
        // After searching all students, if no matches were found for the entered course ID.
        printf("\n[ERROR] This Course ID Can't Be Found at any Student");
        return FIFO_ERROR;  // Return an error code indicating that the course ID wasn't found.
    }
    return FIFO_NO_ERROR;  // Return a success code indicating that the search was successful.
}
```

```c
// Define a function called GET_LENGTH that takes a pointer to a FIFO_Queue_st structure as input.
FIFO_STATUS_EN GET_LENGTH(FIFO_Queue_st *QUEUE) {
    // Check if any of the essential pointers (base, head, tail) of the queue are NULL.
    if (QUEUE->base == NULL || QUEUE->head == NULL || QUEUE->tail == NULL) {
        // If any of the pointers is NULL, print an error message indicating that the queue doesn't exist.
        printf("\n[ERROR] Queue Doesn't Exist, Please Enter Valid FIFO\n\n");
        // Return an error code (FIFO_NULL) to indicate the non-existence of the queue.
        return FIFO_NULL;
    }

    // If the queue exists, print the total number of students (count) currently in the queue.
    printf("\n[INFO] The Total Number of Students is : %d", QUEUE->count);

    // Return a success code (FIFO_NO_ERROR) to indicate that the function executed without errors.
    return FIFO_NO_ERROR;
}
```

```c
// Define a function named FIFO_STATUS_EN that takes a pointer to a FIFO_Queue_st structure as input.
FIFO_STATUS_EN PRINT_LIST(FIFO_Queue_st *QUEUE) {

    // Check if the QUEUE's base, head, or tail pointers are NULL, indicating an invalid queue.
    if (QUEUE->base == NULL || QUEUE->head == NULL || QUEUE->tail == NULL) {
        // Print an error message indicating that the queue doesn't exist.
        printf("\n[ERROR] Queue Doesn't Exist, Please Enter Valid FIFO\n\n");
        // Return a status indicating that the queue is NULL.
        return FIFO_NULL;
    }

    // Print a header indicating the start of printing students' information.
    printf("\n======= Printing All Students Information =======\n");

    // Declare a pointer named Temp to struct Sstudent.
    struct Sstudent* Temp;

    // Initialize Temp to point to the tail of the QUEUE.
    Temp = QUEUE->tail;

    // Iterate over the number of elements in the queue.
    for (int i = 0; i < QUEUE->count; i++) {
        // Call the PRINT_STUDENT function to print the information of the student pointed to by Temp's ID.
        PRINT_STUDENT(QUEUE, Temp->ID);
        // Print two newlines for spacing between student information.
        printf("\n\n");
        // Move Temp to point to the next student's information (assuming a contiguous memory layout).
        Temp++;
    }

    // Return a status indicating that the printing was successful.
    return FIFO_NO_ERROR;
}
```

```c
FIFO_STATUS_EN PRINT_STUDENT(FIFO_Queue_st *QUEUE, int Stud_ID){
    // Function to print student information based on their ID from the FIFO queue.

    // Check if the QUEUE or its components are not initialized.
    if (QUEUE->base == NULL || QUEUE->head == NULL || QUEUE->tail == NULL) {
        printf("[ERROR] Queue Doesn't Exist, Please Enter Valid FIFO\n\n");
        return FIFO_NULL;
    }

    struct Sstudent* Temp;  // Declare a pointer to struct Sstudent to hold temporary student information.
    Temp = QUEUE->tail;     // Initialize Temp with the tail of the queue (last student added).

    // Loop through the students in the queue.
    for(int i = 0; i < QUEUE->count ; i++){
        // Check if the student ID matches the provided Stud_ID.
        if(Temp->ID == Stud_ID){
            printf("\n[INFO] Student ID Is : %d", Temp->ID);            // Print student's ID.
            printf("\n[INFO] Student Name Is : %s %s", Temp->fname, Temp->lname);  // Print student's first and last nam
            printf("\n[INFO] Student GPA Is : %f", Temp->GPA);          // Print student's GPA.

            // Loop through the student's course IDs and print them.
            for(int j = 0; j < Course_Num; j++){
                printf("\n[INFO] Student Course ID Number %d Is : %d", j+1, Temp->CourseID[j]);
            }
        }
        Temp++;  // Move Temp to the next student's information in the queue.
    }
    return FIFO_NO_ERROR;  // Return status indicating successful execution.
}
```

Project Usage

The following sections describe the usage of the project's main functions:

Initialization

The **QUEUE_INIT** function initializes the FIFO queue with the provided arguments. It ensures that the provided pointers are valid and the length is non-zero.

Enqueuing Students

The **ENQUEUE** function adds a student to the FIFO queue. It checks if the queue is full and if the entered student ID is already taken. If successful, it enqueues the student.

Checking Queue Status

The **IS_QUEUE_FULL** and **IS_QUEUE_EMPTY** functions check if the queue is full or empty, respectively.

Adding Students

- **ADD_STUDENT_FROM_FILE** adds students from a predefined dataset to the queue.
- **ADD_STUDENT_MANUALLY** allows users to enter student information manually and adds them to the queue.

Deleting Students

The **DELETE_STUDENT** function removes a student from the queue based on their ID.

Updating Student Information

The **UPDATE_STUDENT** function allows users to update student information, including first name, last name, ID, GPA, or course IDs.

Searching for Students

- **GET_STUDENT_BY_ROLL** searches for students by their ID.
- **GET_STUDENT_BY_FNAME** searches for students by their first name.
- **GET_STUDENT_BY_COURSE** searches for students by a specific course ID.

Getting Queue Length

The **GET_LENGTH** function retrieves and displays the total number of students in the queue.

Printing Student Information

- **PRINT_LIST** displays the information of all students in the queue.
- **PRINT_STUDENT** prints the information of a specific student based on their ID.

Conclusion

The Student Information Management System is a versatile project for managing and organizing student data using a FIFO queue. It provides various functionalities for adding, updating, deleting, and searching for student information. The project can be further extended and customized to meet specific requirements in educational institutions or other settings where student data management is essential.