Project 2

Student Management System Using Queue

GitHub repo:

GitHub - Ziaddelsayed/Embedded_systems_online_diploma

Name: Ziad El-Sayed Ibrahim

Introduction:

-This program implements a simple student management system using a circular FIFO (First-In, First-Out) buffer. The system allows users to manage student records, including adding, removing, searching, and updating student information, all within a fixed memory buffer.

-The core functionality of the program is built around a FIFO data structure, which is ideal for maintaining a sequence of elements (students) in the order they are added, ensuring that the oldest entries are processed or removed first. By utilizing a circular buffer, the program optimizes memory usage, ensuring that space is reused efficiently once students are removed from the system.

-Key features of the program include:

- Manual or File-based Input: Students can be added either manually by entering their details via the console, or automatically by reading from a file.

- Efficient Search Functions: The program allows searching for students by their roll number, first name, or enrolled courses.

- Data Modification: Users can update student details, such as their roll number, name, or courses, ensuring flexibility in maintaining accurate records.

- Data Display: All student records can be displayed, showing comprehensive information, including name, roll number, GPA, and course IDs.

- Deletion and Maintenance: Students can be deleted by roll number, and the FIFO structure automatically shifts remaining records to ensure consistent memory usage.

-This program provides a practical and efficient way to handle a small database of student records, ensuring that memory constraints are respected while offering robust functionality. Through its structured use of the FIFO concept, it demonstrates important principles in memory management and data organization.

-we will talk about some APIs that we used in this program:

```
typedef enum {
    FIFO_no_error,
    FIFO_full,
    FIFO_empty,
    FIFO_Null
}FIFO_Buf_Status;

//FIFO APIs
FIFO_Buf_Status  FIFO_init(FIFO_Buf_t* fifo,struct Sstudent* buff,uint32_t length);
FIFO_Buf_Status  FIFO_enqueue(FIFO_Buf_t* fifo,struct Sstudent* item);
FIFO_Buf_Status  FIFO_dequeue(FIFO_Buf_t* fifo, struct Sstudent* item);
FIFO_Buf_Status  FIFO_IS_FULL(FIFO_Buf_t* fifo);

//Main functions APIs
void add_student_manully(FIFO_Buf_t* fifo);
void add_student_file(FIFO_Buf_t* fifo, const char* file_name);
void find_rl(FIFO_Buf_t* fifo,int roll);
void find_fn(FIFO_Buf_t* fifo, char* fnam);
void find_c(FIFO_Buf_t* fifo, int course_id);
void tot_s(FIFO_Buf_t* fifo);
void del_s(FIFO_Buf_t* fifo, int roll);
void up_s(FIFO_Buf_t* fifo, int roll);
void show_s(FIFO_Buf_t* fifo);
```

1. **FIFO_init**: Initializes a FIFO buffer. It sets the base, head, and tail pointers to the beginning of the buffer and initializes the length and count of the buffer. Returns an error if the buffer is null.

```
#include "fifo.h"
#define DPRINTF(...)        {fflush(stdout);\
        fflush(stdin);\
        printf(__VA_ARGS__);\
        fflush(stdout);\
        fflush(stdin);} //for use printf easily

//FIFO APIs
FIFO_Buf_Status  FIFO_init(FIFO_Buf_t* fifo,struct Sstudent* buff,uint32_t length)
{
    if(buff==NULL)
        return FIFO_Null;

    fifo->base = buff ;
    fifo->head = buff ;
    fifo->tail = buff ;
    fifo->length = length ;
    fifo->count = 0 ;

    return FIFO_no_error;

}
```

2. **FIFO_enqueue**: Adds a new item (student) to the FIFO. It first checks if the FIFO is full or uninitialized, then inserts the new student at the head and adjusts the head pointer to maintain the circular nature of the buffer.

```
FIFO_Buf_Status  FIFO_enqueue(FIFO_Buf_t* fifo,struct Sstudent* item)
{
    if(!fifo->base || !fifo->head || !fifo->tail)
        return FIFO_Null;

    if(FIFO_IS_FULL(fifo)== FIFO_full)
        return FIFO_full;

    *(fifo->head) = *item; // Dereference item to copy the struct
    fifo->count++ ;

    //circler fifo
    if(fifo->head == (fifo->base + (fifo->length - 1)))
        fifo->head = fifo->base;
    else
        fifo->head++;

    return FIFO_no_error;

}
```

3. **FIFO_dequeue**: Removes an item (student) from the FIFO. It checks if the FIFO is empty or uninitialized. If not, it retrieves the student at the tail and adjusts the tail pointer accordingly.

```
FIFO_Buf_Status  FIFO_dequeue(FIFO_Buf_t* fifo, struct Sstudent* item)
{
    if(!fifo->base || !fifo->head || !fifo->tail)
        return FIFO_Null;

    //check fifo
    if(fifo->count == 0)
        return FIFO_empty;

    *item = *(fifo->tail);
    fifo->count-- ;

    //circler fifo
    if(fifo->tail == (fifo->base + (fifo->length - 1)))
        fifo->tail = fifo->base;
    else
        fifo->tail++;

    return FIFO_no_error;


}
```

4. **FIFO_IS_FULL**: Checks if the FIFO buffer is full by comparing the number of elements (count) with the maximum capacity (length). Returns a status indicating whether the FIFO is full or not.

```
FIFO_Buf_Status  FIFO_IS_FULL(FIFO_Buf_t* fifo)
{
    if(!fifo->base || !fifo->head || !fifo->tail)
        return FIFO_Null;

    if(fifo->count == fifo->length)
        return FIFO_full;

    return FIFO_no_error;

}
```

5. **add_student_manually**: Allows the user to manually input the details of a new student (first name, last name, roll number, GPA, course IDs). After gathering the input, it adds the student to the FIFO using FIFO_enqueue.

```
void add_student_manully(FIFO_Buf_t*fifo)
{
    struct Sstudent new_student;
    DPRINTF("Enter the first name:");
    scanf("%s",new_student.student.fname);
    DPRINTF("Enter the last name:");
    scanf("%s",new_student.student.lname);
    DPRINTF("Enter the roll number:");
    scanf("%d",&new_student.student.roll);
    DPRINTF("Enter the GPA:");
    scanf("%f",&new_student.student.GPA);
    DPRINTF("Enter the number of courses:");
    scanf("%d",&new_student.student.num_courses);
    for(int i=0;i<new_student.student.num_courses;i++)
    {
        DPRINTF("Enter Course ID:%d",i+1);
        scanf("%d",&new_student.student.cid[i]);
    }

    if (FIFO_enqueue(fifo, &new_student)==FIFO_no_error){
        DPRINTF("[INFO]Student added successfully!\n");
    }else{
        DPRINTF("[INFO]Error: the Fifo is Full\n");
    }
}
```

6. **add_student_file**: Reads student data from a file and adds each student to the FIFO. Each line in the file should contain a student's details. It parses the input and stores it in the FIFO buffer.

```
void add_student_file(FIFO_Buf_t* fifo, const char* file_name) {
    FILE* file = fopen(file_name, "r");
    if (!file) {
        DPRINTF("[INFO] Error: could not open the file \n");
        return;
    }

    char line[256];
    while (fgets(line, sizeof(line), file)) {
        struct Sstudent new_student;
        memset(new_student.student.cid, 0, sizeof(new_student.student.cid));
        new_student.student.num_courses = 0; // Reset the count

        // Parse the student details from the file
        int num_prases = sscanf(line, "%d %s %s %f",
                &new_student.student.roll,
                new_student.student.fname,
                new_student.student.lname,
                &new_student.student.GPA);

        if (num_prases >= 4) {
            // Read course IDs until the end of the line
            int course_id;
            char* token = strtok(line + (num_prases * sizeof(int)), " "); // Start from the first course ID
            //strtok to read course IDs from the remaining line.

            while (token != NULL && new_student.student.num_courses < 10) {
                course_id = atoi(token);
                new_student.student.cid[new_student.student.num_courses++] = course_id;
                token = strtok(NULL, " ");
            }
            // Add student to FIFO
            if (FIFO_enqueue(fifo, &new_student) == FIFO_no_error) {
                DPRINTF("[INFO] Student %s %s added from file.\n", new_student.student.fname, new_student.student.lname);
            } else {
                DPRINTF("[INFO] Error: FIFO is full, cannot add more students.\n");
                break;
            }
        } else {
            DPRINTF("[INFO] Error: Invalid data format in file.\n");
        }
    }

    fclose(file);
}
```

7. **find_rl**: Searches for a student by roll number in the FIFO. If a match is found, the student's details (name, roll, GPA, courses) are printed. Otherwise, it notifies that the student wasn't found.

```
void find_rl(FIFO_Buf_t* fifo, int roll) {
    struct Sstudent* temp = fifo->base;
    int found = 0;
    for (unsigned int i = 0; i < fifo->count; i++) {
        if (temp->student.roll == roll) {
            DPRINTF("Found Student: %s %s, Roll: %d, GPA: %.2f, Courses: ",
                    temp->student.fname, temp->student.lname,
                    temp->student.roll, temp->student.GPA);
            for (int j = 0; j < temp->student.num_courses; j++) {
                DPRINTF("%d ", temp->student.cid[j]);
            }
            DPRINTF("\n");
            found = 1;
            break;  // Exit the loop once the student is found
        }else{
            temp++;
        }
    }
    if (!found) {
        DPRINTF("[INFO]No student found with roll number: %d\n", roll);
    }
}
```

8. **find_fn**: Searches for a student by their first name. If a match is found, it prints the student's details; otherwise, it indicates that no student with that name was found.

```
void find_fn(FIFO_Buf_t* fifo, char* fname)
{
    struct Sstudent* temp = fifo->base;
    int found = 0;

    for (int i = 0; i < fifo->count; i++)
    {
        if (strcmp(temp->student.fname, fname) == 0)
        {
            DPRINTF("[INFO] Found Student: %s %s, Roll: %d, GPA: %.2f, Courses: ",
                    temp->student.fname, temp->student.lname,
                    temp->student.roll, temp->student.GPA);

            for (int j = 0; j < temp->student.num_courses; j++) {
                DPRINTF("%d ", temp->student.cid[j]);
            }
            DPRINTF("\n");
            found = 1;  // Mark that at least one student was found
        }
        temp++;  // Move to the next student
    }

    if (!found) {
        DPRINTF("[INFO] No student found with first name: %s\n", fname);
    }
}
```

9. **find_c**: Searches for students enrolled in a specific course using the course ID. It loops through all students and checks their course IDs, printing details for any students enrolled in the given course.

```
void find_c(FIFO_Buf_t* fifo, int course_id)
{
    struct Sstudent* temp = fifo->base;
    int found = 0; // Flag to check if any student is found
    DPRINTF("Searching for students enrolled in course ID: %d\n", course_id);

    for (unsigned int i = 0; i < fifo->count; i++) {
        for (int j = 0; j < temp->student.num_courses; j++) {
            if (temp->student.cid[j] == course_id) {
                found = 1; // Set flag to true if a student is found
                // Print the details of the student
                DPRINTF("Student Details:\n");
                DPRINTF("Roll Number: %d\n", temp->student.roll);
                DPRINTF("First Name: %s\n", temp->student.fname);
                DPRINTF("Last Name: %s\n", temp->student.lname);
                DPRINTF("GPA: %.2f\n", temp->student.GPA);
                DPRINTF("Course ID: ");
                for (int k = 0; k < temp->student.num_courses; k++) {
                    DPRINTF("%d ", temp->student.cid[k]);
                }
                DPRINTF("\n----------------------\n");
                break;  // Exit inner loop once a match is found
            }
        }
        temp++;
    }

    if (!found) {
        DPRINTF("No students found enrolled in course ID: %d.\n", course_id);
    }
}
```

10. **tot_s**: Displays the total number of students currently in the FIFO.

```
void tot_s(FIFO_Buf_t* fifo)
{

    DPRINTF("Total number of students: %d\n",fifo->count);

}
```

11. **del_s**: Deletes a student from the FIFO by roll number. It finds the student, then shifts the remaining students to maintain the FIFO structure. The count is decremented, and the tail pointer is updated accordingly.

```
void del_s(FIFO_Buf_t* fifo, int roll) {
    struct Sstudent* temp = fifo->base; // Start from the base of the FIFO
    int found = 0; // Flag to check if the student is found

    // Loop through the FIFO to find the student by roll number
    for (unsigned int i = 0; i < fifo->count; i++) {
        if (temp->student.roll == roll) {
            found = 1; // Student found
            DPRINTF("[INFO] Found Student: %s %s, Roll: %d, GPA: %.2f\n",
                    temp->student.fname, temp->student.lname,
                    temp->student.roll, temp->student.GPA);
            break; // Exit the loop once the student is found
        }
        temp++; // Move to the next student
    }

    if (!found) {
        DPRINTF("[INFO] No student found with roll number: %d\n", roll);
        return; // Exit if no student was found
    }

    // Shift elements to maintain FIFO structure after deletion
    struct Sstudent* student_to_delete = temp; // Pointer to the student to delete

    // Shift all elements after the deleted student one position to the left
    for (unsigned int i = (student_to_delete - fifo->base); i < fifo->count - 1; i++) {
        fifo->base[i] = fifo->base[i + 1]; // Move the next student to the current position
    }


    // Decrease the student count after deletion
    fifo->count--;

    // Update tail pointer
    if (fifo->tail == temp) {
        fifo->tail--; // Move tail back
        if (fifo->tail < fifo->base) {
            fifo->tail = fifo->base + fifo->length - 1; // Wrap around
        }
    }
}
```

12. **up_s**: Updates a student's details by roll number. The user can choose to update the roll number, first name, last

name, or course IDs. For course IDs, the user can either update a specific ID or all of them.

```c
void up_s(FIFO_Buf_t* fifo, int roll) {
    struct Sstudent* temp = fifo->base; // Start from the base of the FIFO
    int found = 0; // Flag to check if the student is found

    // Loop through the FIFO to find the student by roll number
    for (unsigned int i = 0; i < fifo->count; i++) {
        if (temp->student.roll == roll) {
            found = 1; // Student found
            DPRINTF("Updating details for roll number %d\n", roll);

            int choice;
            DPRINTF("What would you like to update?\n");
            DPRINTF("1. Roll Number\n");
            DPRINTF("2. First Name\n");
            DPRINTF("3. Last Name\n");
            DPRINTF("4. Course ID(s)\n");
            DPRINTF("Enter your choice (1-4): ");
            scanf("%d", &choice);

            switch (choice) {
            case 1: {
                int new_roll;
                DPRINTF("Enter new roll number: ");
                scanf("%d", &new_roll);
                temp->student.roll = new_roll; // Update roll number
                DPRINTF("Roll number updated to %d.\n", new_roll);
                break;
            }
            case 2: {
                char new_fname[50];
                DPRINTF("Enter new first name: ");
                scanf("%s", new_fname);
                strcpy(temp->student.fname, new_fname); // Update first name
                DPRINTF("First name updated to %s.\n", new_fname);
                break;
            }

            case 3: {
                char new_lname[50];
                DPRINTF("Enter new last name: ");
                scanf("%s", new_lname);
                strcpy(temp->student.lname, new_lname); // Update last name
                DPRINTF("Last name updated to %s.\n", new_lname);
                break;
            }
            case 4: {
                int update_option;
                DPRINTF("Would you like to update:\n");
                DPRINTF("1. A specific course ID\n");
                DPRINTF("2. All course IDs\n");
                DPRINTF("Enter your choice (1-2): ");
                scanf("%d", &update_option);

                if (update_option == 1) {
                    // Update a specific course ID
                    int course_index, new_cid;
                    DPRINTF("Enter the index (0 to %d) of the course you want to update: ", temp->student.num_courses - 1);
                    scanf("%d", &course_index);

                    if (course_index >= 0 && course_index < temp->student.num_courses) {
                        DPRINTF("Enter new course ID: ");
                        scanf("%d", &new_cid);
                        temp->student.cid[course_index] = new_cid;
                        DPRINTF("[INFO]Course ID at index %d updated to %d.\n", course_index, new_cid);
                    } else {
                        DPRINTF("[INFO]Invalid course index.\n");
                    }
                } else if (update_option == 2) {
                    // Update all course IDs
                    DPRINTF("Enter new course IDs for the student (up to %d):\n", temp->student.num_courses);
                    for (int j = 0; j < temp->student.num_courses; j++) {
                        DPRINTF("Enter new course ID %d: ", j + 1);
                        scanf("%d", &temp->student.cid[j]);
```

```
                    }
                    DPRINTF("All course IDs updated.\n");
                } else {
                    DPRINTF("Invalid choice. Please choose 1 or 2.\n");
                }
                break;
            }
            default:
                DPRINTF("Invalid choice. Please choose between 1 and 4.\n");
                break;
            }
            break; // Exit the loop after updating the student
        }
        temp++; // Move to the next student
    }

    if (!found) {
        DPRINTF("Student with roll number %d not found.\n", roll);
    }
}
```

13. **show_s**: Displays all students currently stored in the FIFO. It loops through the FIFO and prints each student's details (name, roll number, GPA, and course IDs).

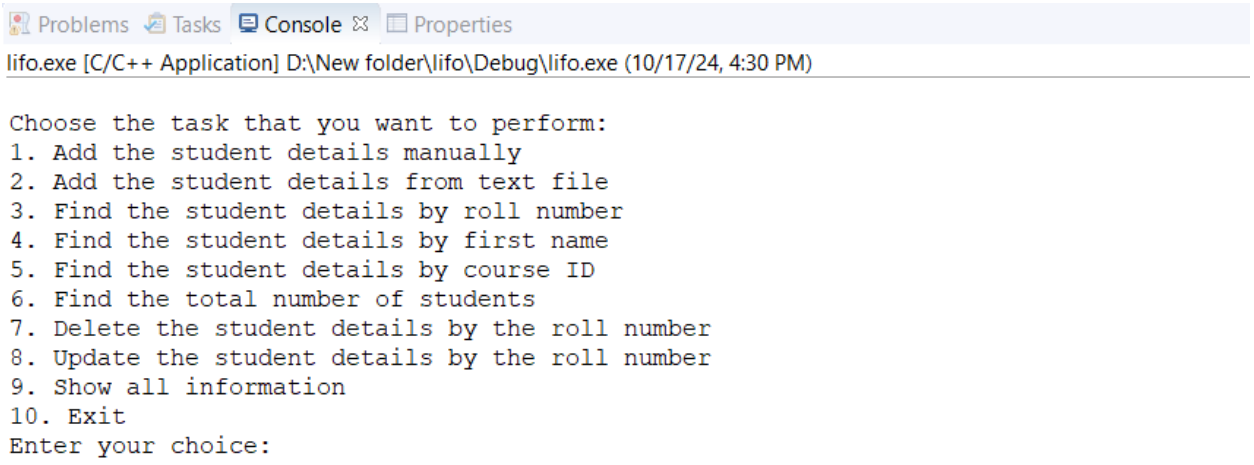
```
void show_s(FIFO_Buf_t* fifo)
{
    struct Sstudent* temp= fifo->base;
    if(fifo->count == 0)
    {
        printf("[INFO]fifo is empyt\n");
    }
    else{
        for(int i=0; i<fifo->count;i++)
        {
            DPRINTF("Student %d: %s %s , ID=%.2f ,CID:",
                    i+1,temp->student.fname,temp->student.lname,temp->student.GPA);
            for(int j=0;j<temp->student.num_courses;j++)
            {
                DPRINTF("%d ",temp->student.cid[j]);

            }
            DPRINTF("\n");
            temp++;
        }
    }


}
```

-Now let’s see how the program works:

-First it will ask you what do you want to do :



- **Add the student details manually:**

```
Enter your choice: 1
Enter the first name:Ziad
Enter the last name:Elsayed
Enter the roll number:1
Enter the GPA:3.1
Enter the number of courses:3
Enter Course ID:1101
Enter Course ID:2102
Enter Course ID:3103
[INFO]Student added successfully!
```

- Let’s enter 2 more students .

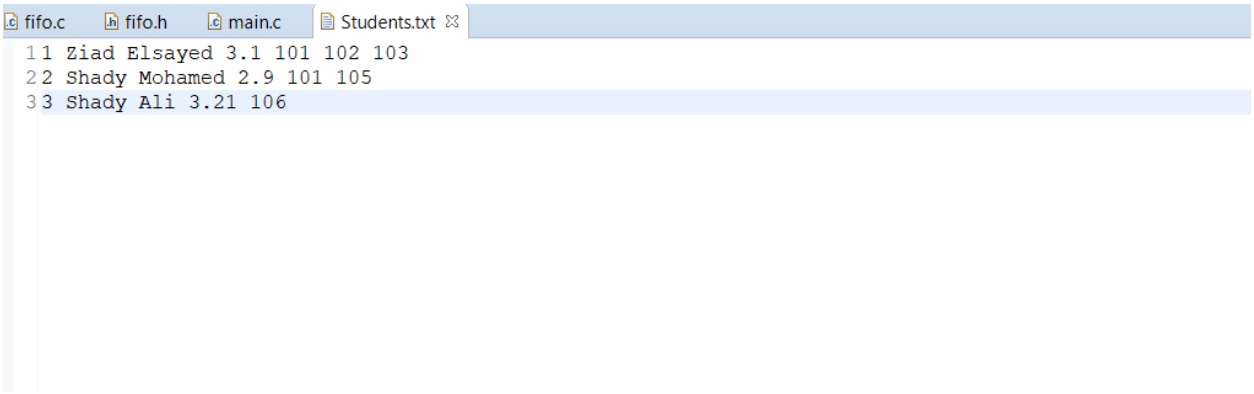
-Now let’s see if the students added successfully by entering 9

To show the students:

- **Show all information:**

```
Enter your choice: 9
Student 1: Ziad Elsayed , ID=3.10 ,CID:101 102 103
Student 2: Shady Mohamed , ID=2.90 ,CID:101 105
Student 3: Shady Ali , ID=3.21 ,CID:106
```

-Now let’s begin a new program and take this students from txt file:



- **Add the student details from text file:**

```
Enter your choice: 2
Enter the file name: src/Students.txt
[INFO] Student Ziad Elsayed added from file.
[INFO] Student Shady Mohamed added from file.
[INFO] Student Shady Ali added from file.

Choose the task that you want to perform:
1. Add the student details manually
2. Add the student details from text file
3. Find the student details by roll number
4. Find the student details by first name
5. Find the student details by course ID
6. Find the total number of students
7. Delete the student details by the roll number
8. Update the student details by the roll number
9. Show all information
10. Exit
Enter your choice: 9
Student 1: Ziad Elsayed , ID=3.10 ,CID:0 101 102 103
Student 2: Shady Mohamed , ID=2.90 ,CID:2 101 105
Student 3: Shady Ali , ID=3.21 ,CID:106
```

-Let’s find the students with roll number 2 (Shady Mohamed).

- **Find the student details by roll number:**

```
--- ----
Enter your choice: 3
Enter the roll number to find: 2
Found Student: Shady Mohamed, Roll: 2, GPA: 2.90, Courses: 101 105
```

-Let’s find Students with First name (we will search for Shady)

It should give us two students.

- **Find the student details by first name:**

```
Enter your choice: 4
Enter the first name to find: Shady
[INFO] Found Student: Shady Mohamed, Roll: 2, GPA: 2.90, Courses: 101 105
[INFO] Found Student: Shady Ali, Roll: 3, GPA: 3.21, Courses: 106
```

-Let’s find Students with course ID (we will search for 101)

It should give us Ziad Elsayed & Shady Mohamed.

- **Find the student details by course:**

```
Enter your choice: 5
Enter the course ID to find: 101
Searching for students enrolled in course ID: 101
Student Details:
Roll Number: 1
First Name: Ziad
Last Name: Elsayed
GPA: 3.10
Course ID: 101 102 103
----------------------
Student Details:
Roll Number: 2
First Name: Shady
Last Name: Mohamed
GPA: 2.90
Course ID: 101 105
----------------------
```

-Let’s See the total number of students (it should be 3).

- **Find the total number of students:**

```
Enter your choice: 6
Total number of students: 3
```

-Let’s delete Student by the roll number (we will delete 3)

It should be Shady Ali.

- **Delete the student details by the roll number:**

```
Enter your choice: 7
Enter the roll number to delete: 3
[INFO] Found Student: Shady Ali, Roll: 3, GPA: 3.21

Choose the task that you want to perform:
1. Add the student details manually
2. Add the student details from text file
3. Find the student details by roll number
4. Find the student details by first name
5. Find the student details by course ID
6. Find the total number of students
7. Delete the student details by the roll number
8. Update the student details by the roll number
9. Show all information
10. Exit
Enter your choice: 9
Student 1: Ziad Elsayed , ID=3.10 ,CID:101 102 103
Student 2: Shady Mohamed , ID=2.90 ,CID:101 105
```

-Let’s update Ziad’s details(roll number 1) (it will ask us what do you want to update) we will update the first course ID from 101 to 106.

- **Update the student details by the roll number:**

```
Enter your choice: 8
Enter the roll number to update: 1
Updating details for roll number 1
What would you like to update?
1. Roll Number
2. First Name
3. Last Name
4. Course ID(s)
Enter your choice (1-4): 4
Would you like to update:
1. A specific course ID
2. All course IDs
Enter your choice (1-2): 1
Enter the index (0 to 2) of the course you want to update: 0
Enter new course ID: 106
[INFO]Course ID at index 0 updated to 106.

Choose the task that you want to perform:
1. Add the student details manually
2. Add the student details from text file
3. Find the student details by roll number
4. Find the student details by first name
5. Find the student details by course ID
6. Find the total number of students
7. Delete the student details by the roll number
8. Update the student details by the roll number
9. Show all information
10. Exit
Enter your choice: 9
Student 1: Ziad Elsayed , ID=3.10 ,CID:106 102 103
Student 2: Shady Mohamed , ID=2.90 ,CID:101 105
```

-That’s all we want you to know about our program thanks for reading this , may we will meet again inshallah!