Course registration system

Team Members

* Suliaiman Alabdullah
* Omar Almatroodi

This project focused on managing students, course, and registration data through reading and writing to text files.

To start, we clearly outlined the project's requirements and identified the steps needed to accomplish the objectives. This included defining the structure of our program and the functions required for processing student, course, and registration data. After brainstorming and discussing our ideas, we agreed on the overall design and architecture. This planning process done by both members of the team.

Program Structure:

Error Handling: An error message is displayed if the required text files aren't available in the project folder. This ensures reliable data access.

Reusable Functions: Functions such as 'read list of students' and 'read list of courses' are used across different menu options, reducing code complexity and improving maintainability.

Data Processing: Data is first read into Python lists or dictionaries before any changes are made.

Continuous Menu: The main menu is kept active within a while loop, allowing continuous operation until the program ends.

Suliaiman was responsible for implementing the file-handling functions. He created code to read, write, and update the data files for students, courses, and registrations. His primary focus was on ensuring that the program could handle input and output operations reliably while maintaining data integrity.

Meanwhile, Omar took charge of implementing the user interface and menu system. He created the functions that presented a user-friendly menu to guide users through their choices. He also worked on the functions that handled each menu option, ensuring that data was processed correctly and error messages were provided when appropriate.

We frequently communicated our progress and helped each other troubleshoot challenges we encountered. For instance, when Omar encountered a problem with listing registered courses by student ID, Suliaiman provided assistance and suggestions that helped fix the issue. Similarly, Omar helped Suliaiman refine the functions that validated input data.

By working together and sharing our strengths, we solved the problem effectively. We carefully tested and validated the program to ensure it met all requirements and was ready for use. Our collaboration ensured that each of us contributed as equally as possible, and the final result is a testament to our teamwork. Any differences in our contributions were addressed through constant communication and by helping each other overcome obstacles.

Contribution to the project from team members

**Suliaiman Alabdullah:**

* Implemented core file-handling functions, including read, write, and update.
* Designed the main loop to maintain the continuous menu flow.
* Developed functions like view\_list\_of\_courses, search\_student\_by\_id, update\_course\_name, and list\_registered\_courses.
* Handled file initialization to ensure the presence of all necessary text files.
* Provided validation logic for input data to ensure integrity.

focused on the core logic and file handling functions for the project. He wrote the code that manages the reading, writing, and updating of data files for students, courses, and registrations. His work included ensuring proper validation for inputs and maintaining data integrity. Suliaiman was also responsible for implementing the file initialization function, which ensures that all required files are available. Additionally, he assisted with troubleshooting challenges, particularly around data handling.

**Omar Almatroodi**

* Built the user interface, including the menu system that guides users through their choices.
* Developed functions such as view\_list\_of\_students, add\_new\_student, delete\_course, register\_course, and add\_new\_course.
* Ensured each menu option interacted correctly with the data files and displayed error messages when necessary.
* Assisted Suliaiman with troubleshooting and refining functions related to input validation.

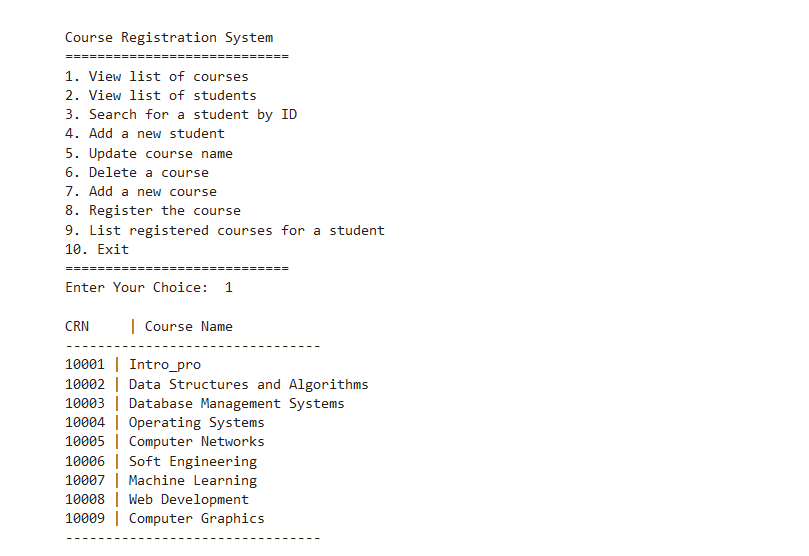
handled the user interface and control flow, designing the main menu system that allows users to navigate through the program seamlessly. He implemented functions for each menu option to ensure they interact with the data correctly, providing error messages when necessary. Omar's work included functions for searching, listing, and registering courses, all of which required managing the menu input and coordinating actions. He also contributed to debugging by refining functions related to student and course validation.

**Here's the description of each function, keeping the same format:**

1. **ask\_user\_input**:  
   This function displays a user-friendly menu of options and prompts the user to enter their choice. It returns the input as a string to the main loop for further processing.
2. **initialize\_files**:  
   Calls ensure\_file\_exists for each data file (students, courses, registrations) to guarantee that all necessary files exist at the beginning of the program.
3. **read\_list\_of\_courses**:  
   Reads the list of courses from the courses file and returns them as a list of pairs [CRN, course name]. Each course is processed line by line to extract its information.
4. **read\_list\_of\_students**:  
   Similar to read\_list\_of\_courses, this function reads the list of students from the students file and returns them as a list of pairs [student ID, student name].
5. **read\_list\_of\_registered\_courses**:  
   Retrieves the list of registered courses by reading the data from the registrations file. It returns a list of pairs [student ID, CRN].
6. **view\_list\_of\_courses**:  
   Calls read\_list\_of\_courses and prints the list of courses in a neatly formatted table. It displays the CRN and course name for each course.
7. **view\_list\_of\_students**:  
   Calls read\_list\_of\_students and prints the student data in a formatted table. It shows the student ID and student name for each student.
8. **search\_for\_a\_student\_by\_ID**:  
   Prompts the user to input a student ID and searches for it in the list of students. If found, it prints the student's name and lists the courses the student has registered for. Otherwise, it displays an error message.
9. **is\_valid\_student\_id**:  
   Checks if a student ID is a 7-digit numeric value. It ensures that the input is composed entirely of digits and is exactly seven characters long.
10. **write\_new\_student**:  
    Takes a new student ID and name as input and writes these details into the students file in an append mode. This allows new students to be added to the existing file without overwriting existing data.
11. **add\_a\_new\_student**:  
    Prompts the user to input a 7-digit student ID and a name. It verifies that the ID is unique and the name is valid, and then writes the new student's data using ‘write\_new\_student’.
12. **write\_courses**:  
    Takes a dictionary of courses and writes them into the courses file, overwriting the previous data.
13. **read\_courses**:  
    Reads the courses file and returns a dictionary where the key is the CRN and the value is the course name. This allows for quick lookups and easy modification.
14. **write\_updated\_courses**:  
    Uses the data structure returned by ‘read\_courses’ to write an updated list of courses back into the courses file.
15. **update\_course\_name**:  
    Prompts the user for a specific CRN and checks if it exists in the courses file. If it does, the function prompts the user for a new course name, updates the dictionary, and writes the updated data back using ‘write\_updated\_courses’.
16. **write\_registrations**:  
    Takes a list of student ID and CRN pairs and writes them to the registrations file, overwriting any existing data.
17. **delete\_registrations\_for\_course**:  
    Removes all registrations associated with a particular CRN. It filters out the relevant entries from the registrations file and returns the updated list.
18. **delete\_a\_course**:  
    Prompts the user for a CRN to delete, checks if it exists in the courses dictionary, and removes it if confirmed by the user. It updates the courses file using ‘write\_updated\_courses’ and removes associated registrations using ‘delete\_registrations\_for\_course’.
19. **is\_valid\_crn**:  
    Verifies that a CRN is a 5-digit integer and is not already present in a list of existing CRNs. Ensures that new CRNs are unique and correctly formatted.
20. **write\_new\_course**:  
    Writes a new course to the courses file in append mode. This allows adding new courses without modifying existing data.
21. **is\_valid\_course\_name**:  
    Ensures that a course name contains only letters and spaces and is not empty after removing extra whitespace.
22. **add\_new\_course**:  
    Prompts the user for a new 5-digit unique CRN and a valid course name. It checks for duplicates and valid inputs, then writes the new course using’ write\_new\_course’.
23. **read\_students**:  
    Reads the students file and returns a dictionary where the key is the student ID and the value is the student name. This allows for easy lookup and validation.
24. **write\_new\_registration**:  
    Takes a student ID and a CRN and appends this new registration to the registrations file.
25. **register\_course**:  
    Prompts the user for a student ID and a course CRN, verifies that both exist, and writes the new registration using ‘write\_new\_registratio’n.
26. **read\_registrations**:  
    Reads the registrations file and returns a dictionary where the key is the student ID and the value is a list of CRNs.
27. **list\_registered\_courses\_for\_student**:  
    Prompts the user for a student ID, retrieves the list of courses they have registered for, and displays them in a readable format. If no courses are found for the student ID, it displays an appropriate error message.

**Screen shots of the running code**

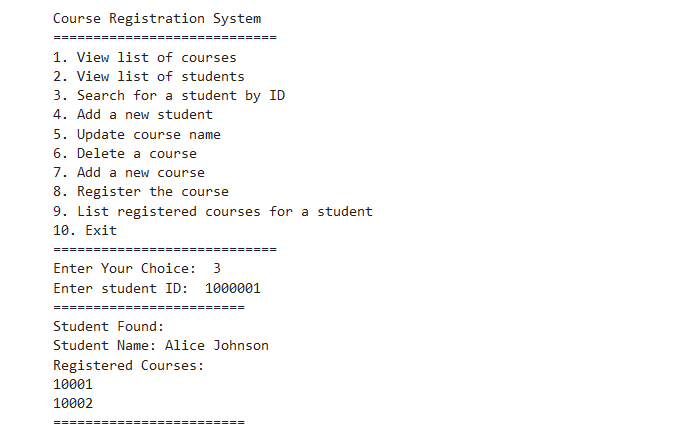
1. View list of courses



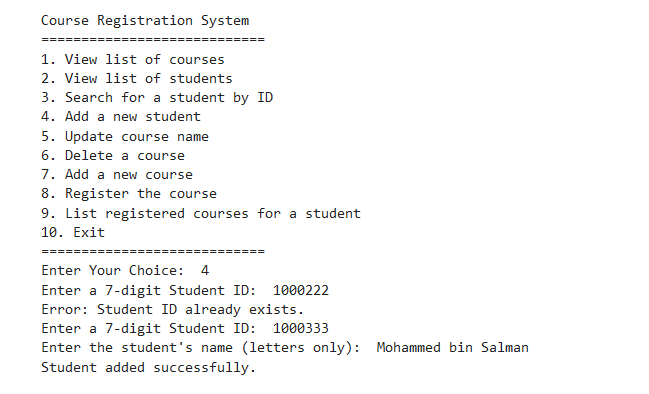
2. View list of students



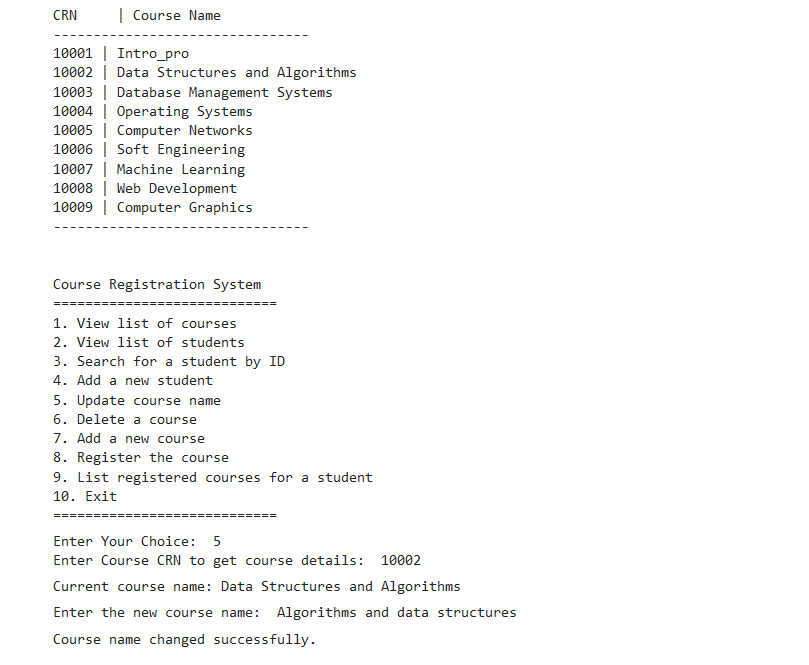
3. Search for a student by ID



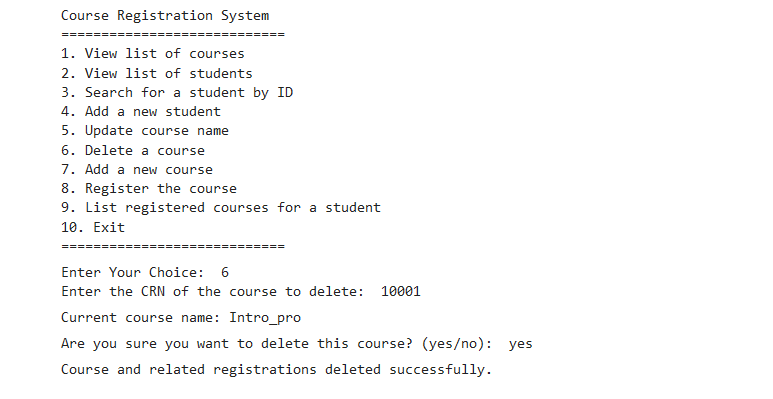
4. Add a new student



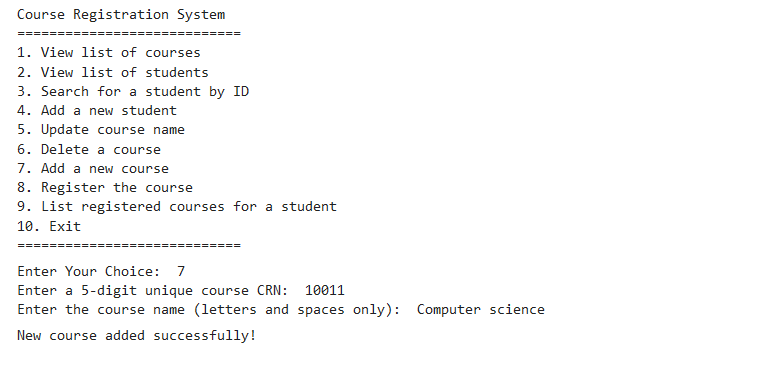
5. Update course name



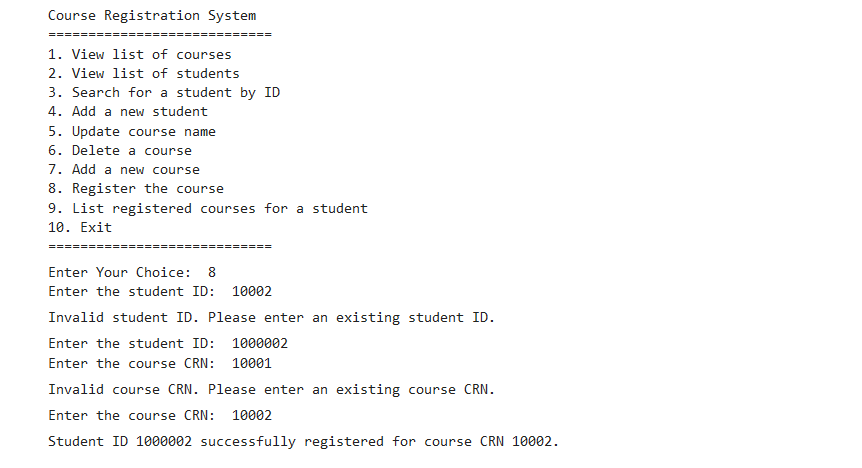
6. Delete a course



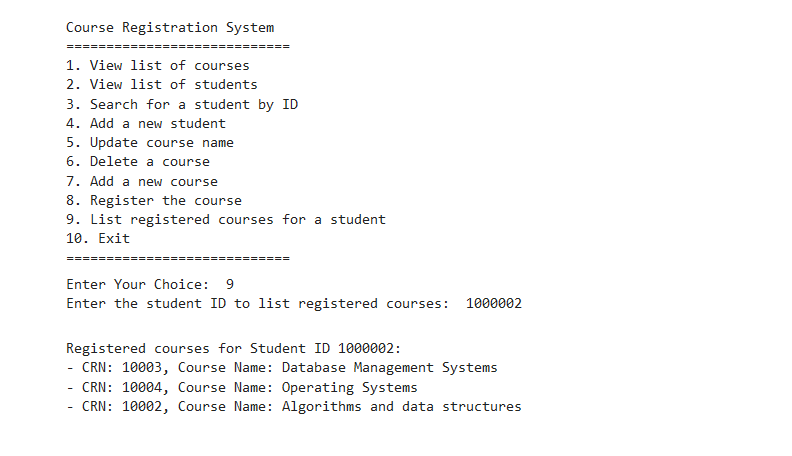
7. Add a new course



8. Register the course



9. List registered courses for a student



10. Exit

