* **Subject: PRF192 - Programming Fundamental with C**

**Assignment**

**Objectives:**

* To develop a student management system for storing, searching, and sorting student data.
* To practice working with text files for data storage and retrieval.
* To implement structured programming with user-defined functions for modularity.
* To apply sorting algorithms and searching techniques in a real-world scenario.

**Problem Description:**

Create a program in C to manage student information in a text file. The program should:

1. Allow users to **add new students** to the file.
2. Enable users to **search for students** by their ID or last name.
3. Display the list of students **sorted in ascending order by last name**.

**Situation Description:**

A university wants a simple student management system for storing student records in a text file. The system should support:

* Adding new student records with details like ID, first name, last name, and GPA.
* Searching for students by ID or last name to quickly retrieve details.
* Displaying a sorted list of all students by their last name for easy reference.

This system helps manage and access student records effectively without requiring a database.

**Syntax Use in the Problem:**

1. **File Handling**: Used for reading from and writing to a text file.
2. **Functions**: Modularize tasks like adding, searching, sorting, and displaying data.
3. **String Manipulation**: Used for handling student names and performing searches.
4. **Loops**: Iterate through student records.
5. **Conditional Statements**: Implement logic for searching and sorting.

**Specific Requirements:**

1. **Data Format**: Each student record should include:

* ID: Integer (e.g., 101)
* First Name: String (e.g., John)
* Last Name: String (e.g., Doe)
* GPA: Float (e.g., 3.75)

1. **Functions to Implement**:

* void addStudent(const char \*filename);
* void displayStudents(const char \*filename);
* void searchStudentById(const char \*filename, int id);
* void searchStudentByLastName(const char \*filename, const char \*lastName);
* void sortStudentsByLastName(const char \*filename);

1. **File Format**: The student data should be stored in a plain text file (**students.txt**) with the following format for each record:

|  |
| --- |
| ID,FirstName,LastName,GPA |

1. **Menu Options**:

* Add a new student.
* Search for a student by ID.
* Search for a student by last name.
* Display all students sorted by last name.

1. **Output Requirements**:

* Properly formatted table of student data when displaying.
* Meaningful messages for successful or unsuccessful search operations.

**Detailed Evaluation Criteria:**

**Stage 1: Basic Functionality (2 points)**

* **Focus:** Implementing fundamental features: adding and displaying students.
* **Criteria:**
  + Correct implementation of the addStudent function.
  + Accurate reading and displaying of all records in displayStudents.
  + Proper file operations: opening, writing, reading, and closing files without errors.

**Stage 2: Search Operations (2.5 points)**

* **Focus:** Searching students by ID and last name.
* **Criteria:**
  + searchStudentById accurately finds and displays student details based on the provided ID.
  + searchStudentByLastName handles searching by last name and displays matching records.
  + Correct handling of cases where no matching records are found.

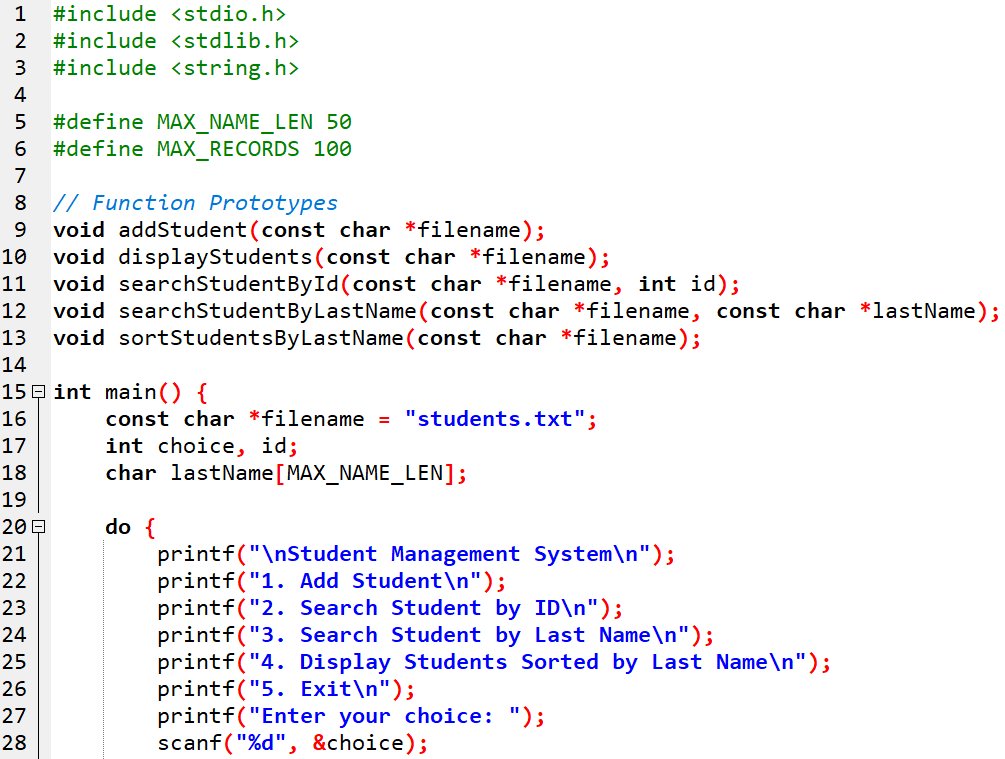
**Stage 3: Sorting Functionality (3 points)**

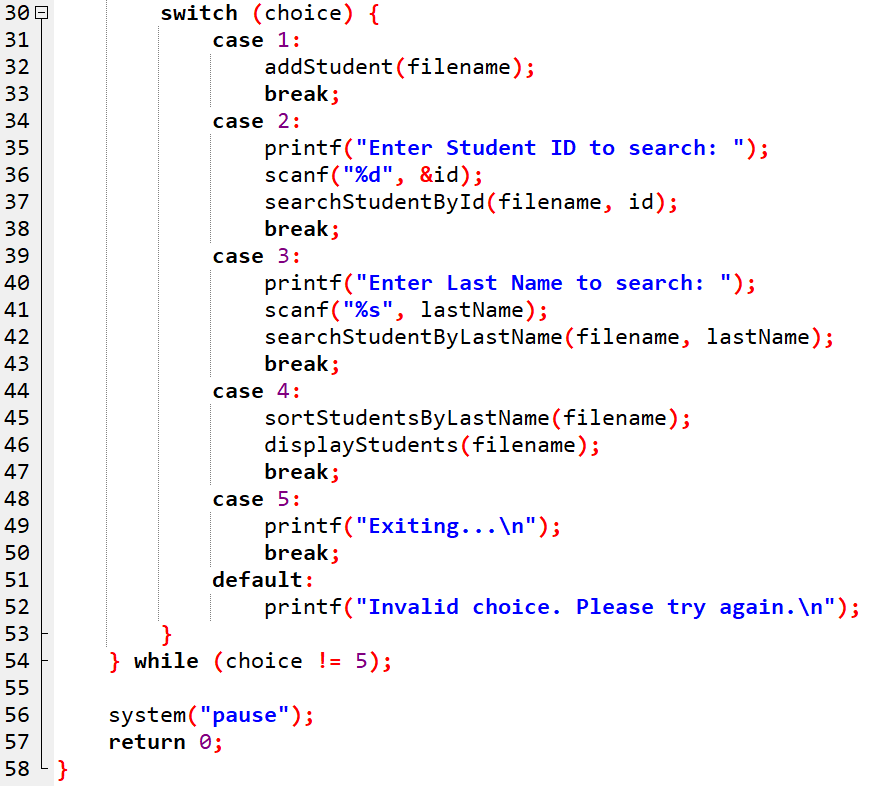
* **Focus:** Sorting student records by last name.
* **Criteria:**
  + Correct implementation of sorting logic in sortStudentsByLastName.
  + Sorted records are written back to the file in the correct order.
  + Bubble sort algorithm works efficiently for the given data structure.
  + Proper integration with displayStudents to confirm sorting functionality.

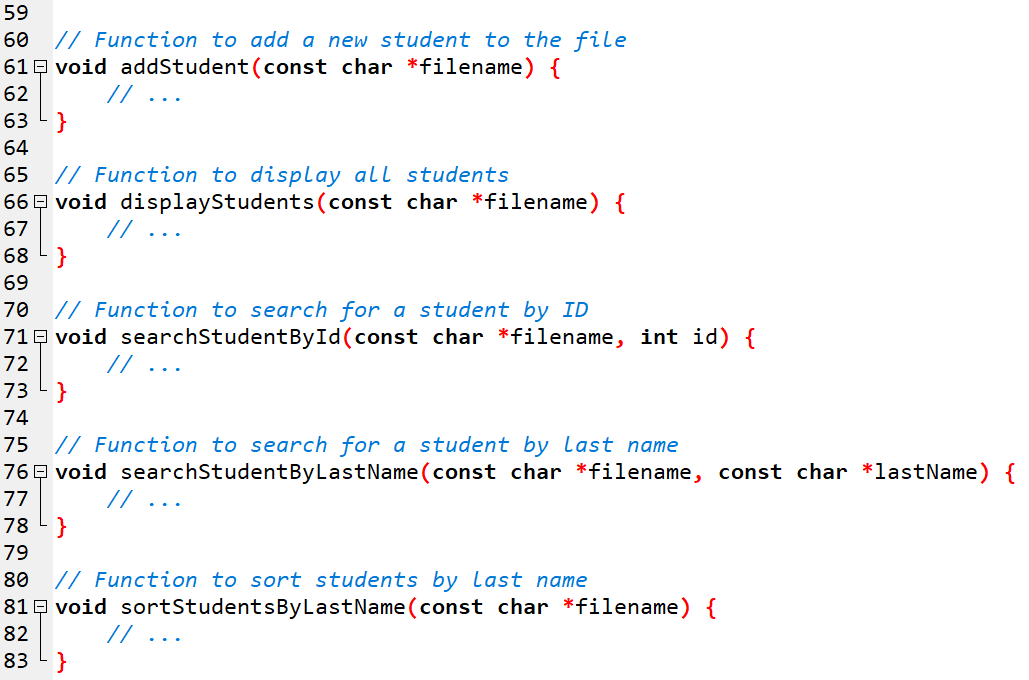
**Stage 4: Menu and Error Handling (2.5 points)**

* **Focus:** Creating a robust and interactive menu-driven program.
* **Criteria:**
  + The menu allows seamless navigation through options using do-while and switch.
  + Invalid inputs are handled gracefully, with appropriate messages for the user.
  + The program does not crash on file errors or invalid data.

**Sample Code:**







**Output Sample:**

|  |
| --- |
| -- Student Management System --  1. Add Student  2. Search Student by ID  3. Search Student by Last Name  4. Display Students Sorted by Last Name  5. Exit  Enter your choice: 1  Enter Student ID: 101  Enter First Name: John  Enter Last Name: Doe  Enter GPA: 3.75  Student added successfully.  -- Student Management System --  1. Add Student  2. Search Student by ID  3. Search Student by Last Name  4. Display Students Sorted by Last Name  5. Exit  Enter your choice: 2  Enter Student ID to search: 101  Student Found:  ----------------------------------  ID: 101  First Name: John  Last Name: Doe  GPA: 3.75  -- Student Management System --  1. Add Student  2. Search Student by ID  3. Search Student by Last Name  4. Display Students Sorted by Last Name  5. Exit  Enter your choice: 3  Enter Last Name to search: Smith  Student Found:  ----------------------------------  ID: 102  First Name: Alice  Last Name: Smith  GPA: 3.90  -- Student Management System --  1. Add Student  2. Search Student by ID  3. Search Student by Last Name  4. Display Students Sorted by Last Name  5. Exit  Enter your choice: 4  Students Sorted by Last Name:  -------------------------------------------------  ID First Name Last Name GPA  -------------------------------------------------  102 Alice Smith 3.90  101 John Doe 3.75  103 Bob Allen 3.50  -- Student Management System --  1. Add Student  2. Search Student by ID  3. Search Student by Last Name  4. Display Students Sorted by Last Name  5. Exit  Enter your choice: 5  Exiting... |

**-- End --**