****

**FATIMA JINNAH WOMEN UNIVERSTIY**

**Programming Fundamentals fall 2024-25**

**Report writing of student information system**

# SUBMITTED TO:

# Mr. ADEEL KHALID

**Lab Engineer**

**SUBMITTED BY:**

|  |  |
| --- | --- |
| **GROUP MEMBERS** | **REGISTRATION NUMBER** |
| **SYEDA ALIA BATOOL** | **FA24B1-CS-087** |
| **ALEESHA MUSLIM** | **FA24B1-CS-064** |
| **ALIZA SEHAR** | **FA24B1-CS-012** |
| **RAHEELA NISAR** | **FA24B1-CS-080** |

Contents

[**STUDENT INFORMATION SYSTEM** 3](#_Toc187963067)

[**1. Overview** 3](#_Toc187963068)

[Introduction 3](#_Toc187963069)

[**2. Features and Functionalities** 3](#_Toc187963070)

[**2.1 Add Student Details** 3](#_Toc187963071)

[4](#_Toc187963072)

[**2.2 View Student Details** 6](#_Toc187963073)

[**2.3 Find Student by Roll Number** 7](#_Toc187963074)

[**2.4 Update Student Details** 8](#_Toc187963075)

[**2.5 Delete Student Details** 9](#_Toc187963076)

[**2.6 Exit Program** 9](#_Toc187963077)

[**3. Workflow** 10](#_Toc187963078)

[**3.1 Program Flow** 10](#_Toc187963079)

[**3.2 Function Execution** 10](#_Toc187963080)

[**4. Implementation** 11](#_Toc187963081)

[**4.1 Data Structure** 11](#_Toc187963082)

[**4.2 Key Functions** 11](#_Toc187963083)

[**5. Error Handling** 11](#_Toc187963084)

[**7.** **Flowchart** 12](#_Toc187963085)

[**Description of Flowchart** 12](#_Toc187963086)

[**7. Program Strengths** 17](#_Toc187963087)

[**8. Limitations and Future Improvements** 17](#_Toc187963088)

[8. Conclusion 18](#_Toc187963089)

**STUDENT INFORMATION SYSTEM**

**1. Overview**

The **Student Information System (SIS)** is a console-based application developed in the C programming language, designed for managing student records. The application enables administrators to perform a variety of operations on student data, such as adding, updating, viewing, deleting, and finding student details. Furthermore, it supports storing and loading student data from a file, ensuring that the system retains student records between program runs.

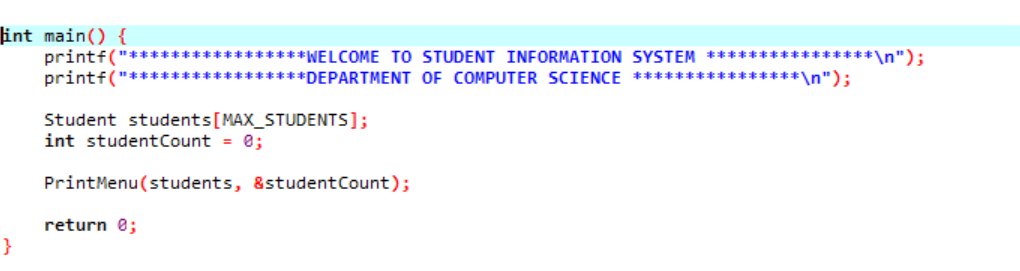
This report documents the system’s purpose, functionality, structure, and potential for future enhancements.

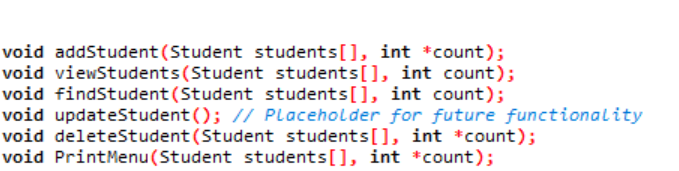
### ****Introduction****

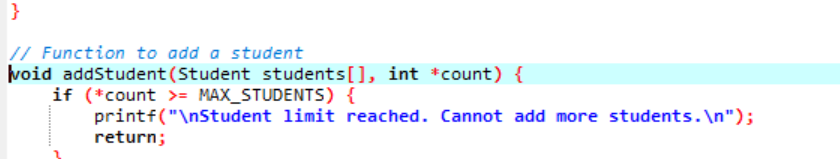
The Student Information System (SIS) serves as a vital tool for organizing and managing student data in an educational institution. By providing a streamlined interface, it enables users to perform essential operations, such as adding new students, updating their information, viewing existing records, and deleting outdated or incorrect entries. Designed with scalability and simplicity in mind, the SIS ensures that users can handle records effortlessly while minimizing errors. This report outlines the system's functionalities, workflow, and implementation details, alongside identifying potential areas for enhancement.

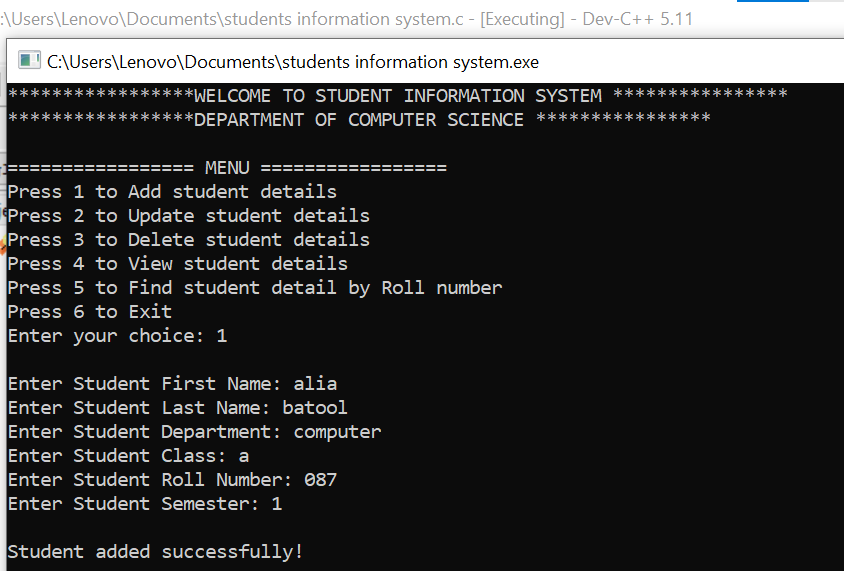
**2. Features and Functionalities**

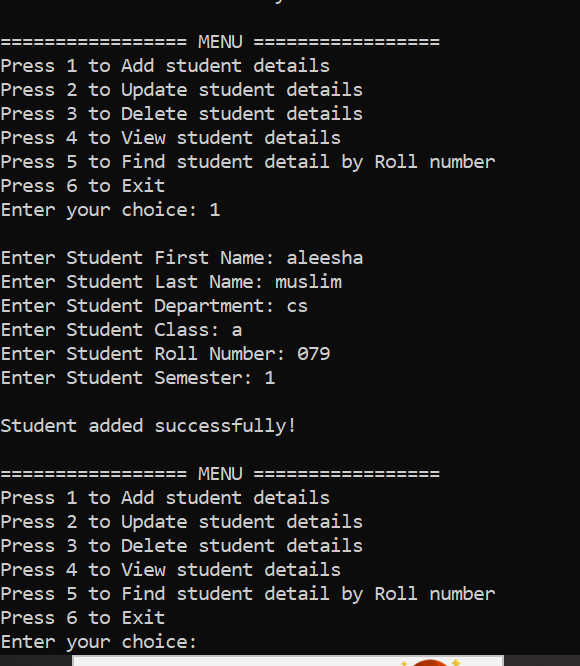
**2.1 Add Student Details**

****

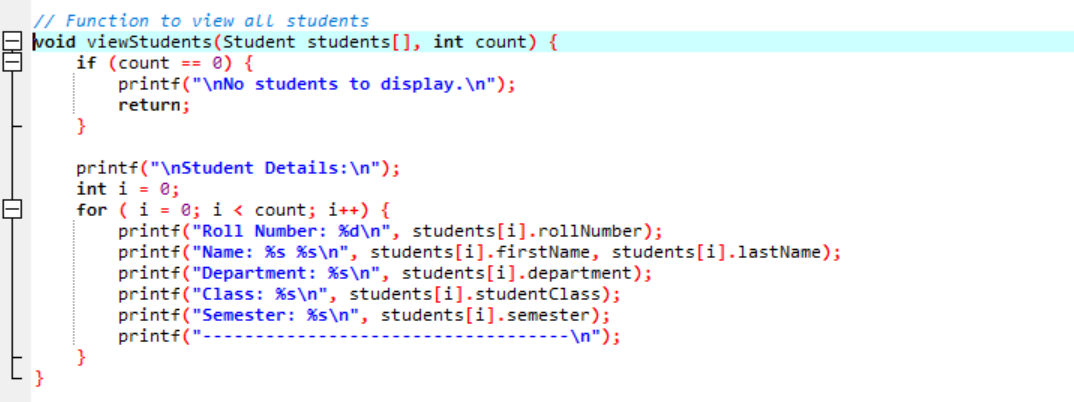
* Allows the user to input details like:
  + First Name
  + Last Name
  + Department
  + Class
  + Roll Number
  + Semester
  + 
* Prevents adding more students if the limit of 100 is reached.

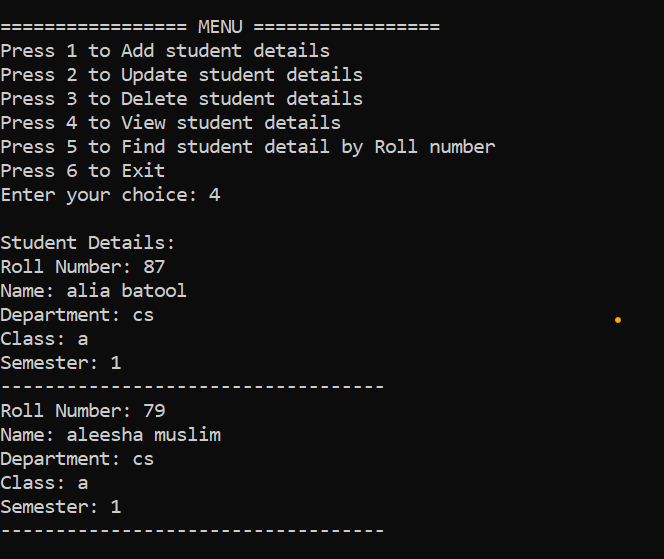


* Displays a confirmation message upon successful addition.
* 

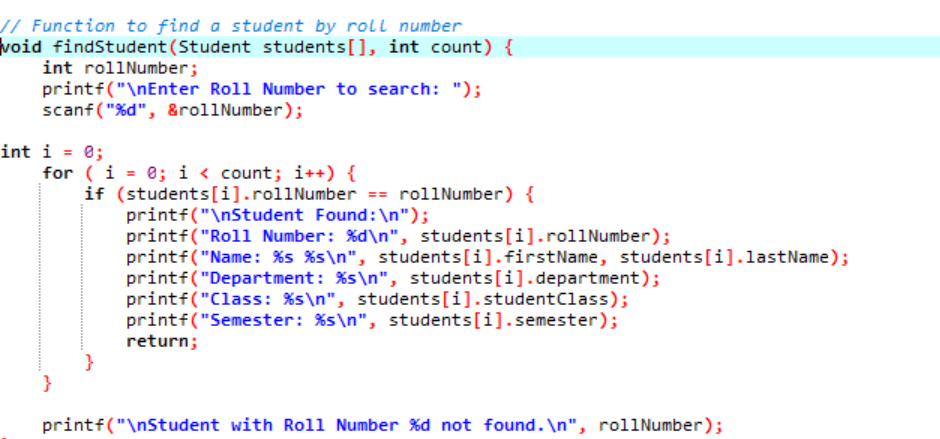


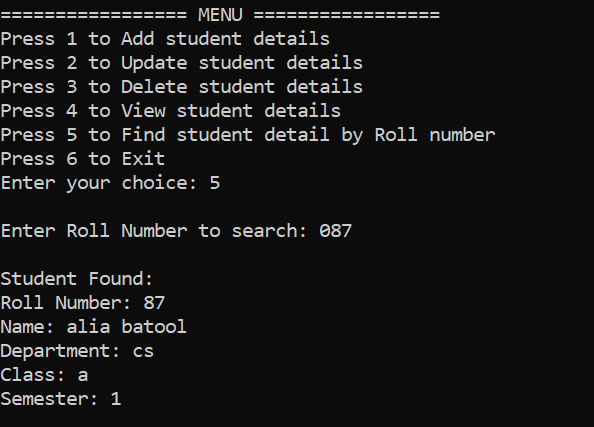
**2.2 View Student Details**

* Lists all stored student records, including:
  + Roll Number
  + Full Name
  + Department
  + Class
  + Semester
* 
* Handles scenarios where no students exist and displays an appropriate message.



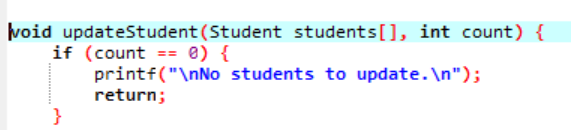
**2.3 Find Student by Roll Number**

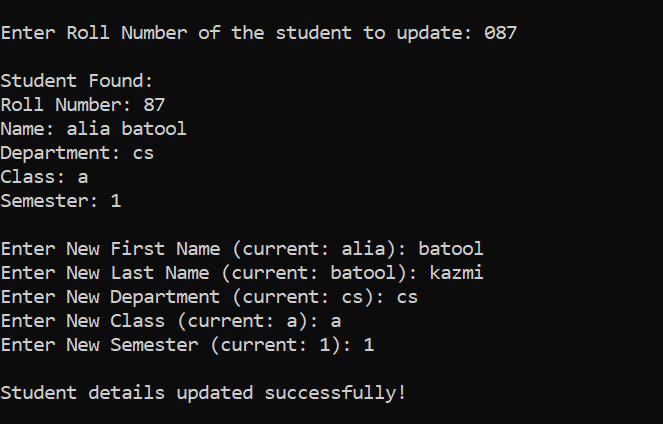
* Searches for a student using their unique roll number.
* If the roll number is found, displays the student’s complete details.
* If the roll number does not exist, an error message is shown.
* 



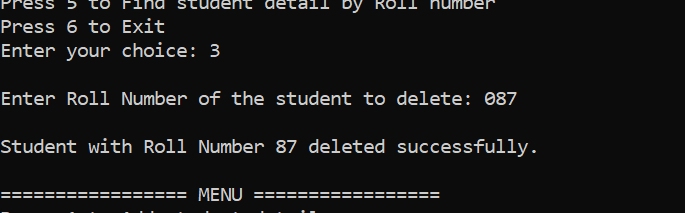
**2.4 Update Student Details**

* Locates a student record by roll number.
* Allows the user to update any of the following fields:
  + First Name
  + Last Name
  + Department
  + Class
  + Semester



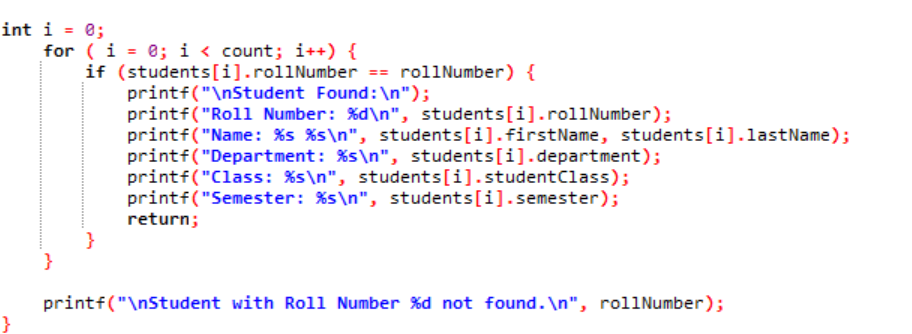
* Verifies the existence of the roll number before proceeding.
* 

**2.5 Delete Student Details**

* Removes a student record using the roll number.
* Ensures that all subsequent records are shifted correctly to maintain data consistency.
* Reduces the total student count after deletion.
* 

**2.6 Exit Program**

* Exits the system with a farewell message.



**3. Workflow**

**3.1 Program Flow**

1. **Welcome Message**:
   * Displays a greeting and initializes the student array and counter.
2. **Main Menu**:
   * Presents the user with options to perform various actions (Add, View, Update, Delete, Find, Exit).
3. **Input Validation**:
   * Ensures the user selects a valid menu option.
   * If invalid input is detected, prompts the user to try again.
4. **Loop Execution**:
   * Continues displaying the menu and executing actions until the user opts to exit.

**3.2 Function Execution**

Each menu option corresponds to a specific function:

* **Add Student**: Inputs details and increments the student count.
* **View Students**: Iterates through the array and displays all student records.
* **Find Student**: Searches by roll number and outputs matching details.
* **Update Student**: Edits fields of an existing student record.
* **Delete Student**: Removes a record, shifts subsequent entries, and updates the count.
* **Exit Program**: Terminates the loop and ends the program.

**4. Implementation**

**4.1 Data Structure**

The program uses a struct to store student details:

typedef struct {

char firstName[50];

char lastName[50];

char department[100];

char studentClass[20];

char semester[20];

int rollNumber;

} Student;

This structure ensures efficient storage and retrieval of student data.

**4.2 Key Functions**

1. **addStudent**: Adds a new student to the array.
2. **viewStudents**: Displays all existing records.
3. **findStudent**: Searches for a student by roll number.
4. **updateStudent**: Updates fields for a specific student.
5. **deleteStudent**: Deletes a record and maintains array integrity.
6. **PrintMenu**: Handles user interactions through the menu system.

**5. Error Handling**

* Prevents adding more than 100 students.
* Displays appropriate messages for invalid roll numbers during search, update, or delete operations.
* Ensures graceful handling of empty records during view or update operations.
* Validates menu options and prompts re-entry for invalid choices.

1. **Flowchart**

**Description of Flowchart**

1. **Start**:
   * Display a welcome message.
   * Initialize an array to store student records and set the student counter to 0.
2. **Main Menu**:
   * Display menu options (1–6).
   * Based on the user’s selection:
     + Option 1: Add a new student (if below limit).
     + Option 2: Update student details (if records exist).
     + Option 3: Delete a student (if records exist).
     + Option 4: View all students (if records exist).
     + Option 5: Find a student by roll number.
     + Option 6: Exit the program.
3. **Repeat Menu**:
   * If an invalid option is entered, display an error and loop back to the menu.
4. **Exit**:
   * Display a goodbye message and terminate the program.

**FLOWCHART**

**File Operations in the Program**

The file-related functionality is implemented in the addStudent function. The file operations include:

* Opening a file for appending ("a" mode).
* Writing student details into the file.
* Properly closing the file after writing.

**2. Location of the File**

The program writes the data to the file path:

makefile

CopyEdit

D:\\p\\prgram.txt

This assumes the directory D:\p exists. If the directory does not exist, the program will fail to open the file. The file mode ("a") ensures:

* Data is appended to the file without overwriting existing content.
* A new file is created if it does not already exist.

**3. Implementation in the addStudent Function**

Here’s how the file handling is performed in the addStudent function:

1. **File Opening**

FILE \*fptr = fopen("D:\\p\\prgram.txt", "a");

The file is opened in append mode ("a") to ensure new data is added to the file without overwriting existing content.

1. **Data Writing** After taking user input for student details, the program writes each field to the file using fprintf:

fprintf(fptr, "\n");

fprintf(fptr, "%s", students[\*count].firstName);

fprintf(fptr, "\n");

fprintf(fptr, "%s", students[\*count].lastName);

// Similar lines for other fields...

The details include:

* + First Name
  + Last Name
  + Department
  + Class
  + Roll Number
  + Semester

The program appends a separator (...................Student Added....................) to make the data visually clear in the file:

c

fprintf(fptr, "\n");

fprintf(fptr, "...................Student Added....................");

1. **File Closing** The file is closed using:

fclose(fptr);

Closing the file ensures all data is written to disk and prevents resource leaks.

**4. Suggestions for Improvement**

Here are some recommendations to improve the file handling:

1. **Error Handling** The program does not currently handle cases where the file cannot be opened. Add error handling to notify the user if the file operation fails:

if (fptr == NULL) {

printf("Error: Could not open file.\n");

return;

}

1. **Formatted Data Storage** Instead of writing each field on a new line, consider using a single formatted line for each student. For example:

fprintf(fptr, "%d,%s,%s,%s,%s,%s\n",

students[\*count].rollNumber,

students[\*count].firstName,

students[\*count].lastName,

students[\*count].department,

students[\*count].studentClass,

students[\*count].semester);

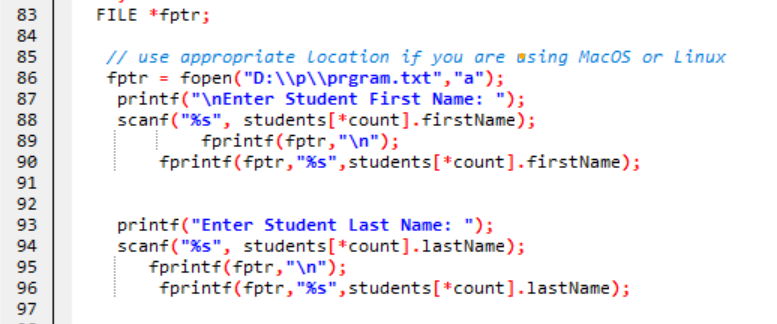
This makes the file more structured and easier to parse if needed.

1. **File Path as a Constant** Use a macro or constant for the file path so that it can be modified easily:

#define FILE\_PATH "D:\\p\\prgram.txt"

FILE \*fptr = fopen(FILE\_PATH, "a");

1. **Read from the File** Add functionality to read student details from the file when the program starts. This would allow the program to persist student data across multiple executions.



**5. Adding Read Functionality**

A readFromFile function could be implemented to load data from the file into the students array at the start of the program. Example:

void readFromFile(Student students[], int \*count) {

FILE \*fptr = fopen("D:\\p\\prgram.txt", "r");

if (fptr == NULL) {

printf("File not found. Starting with an empty list.\n");

return;

}

while (fscanf(fptr, "%d,%49[^,],%49[^,],%99[^,],%19[^,],%19[^\n]\n",

&students[\*count].rollNumber,

students[\*count].firstName,

students[\*count].lastName,

students[\*count].department,

students[\*count].studentClass,

students[\*count].semester) == 6) {

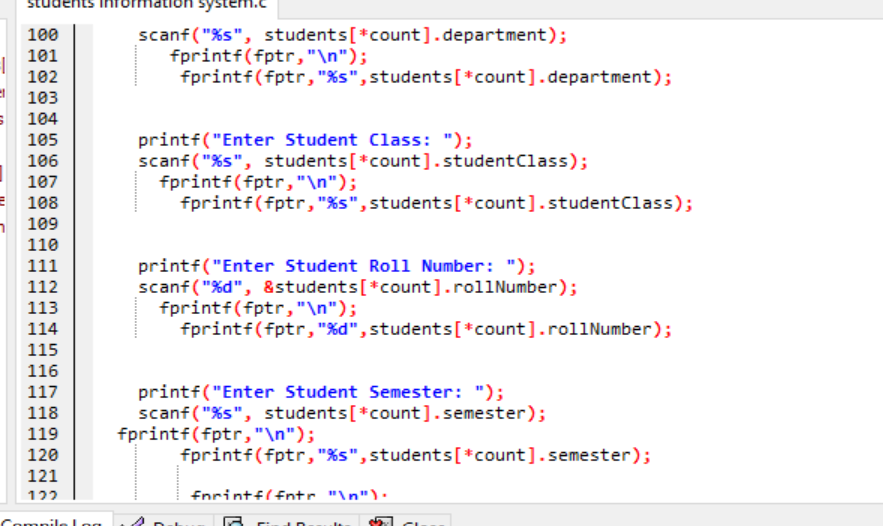
(\*count)++;

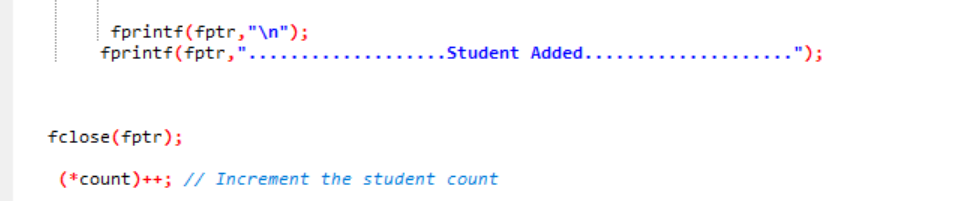
}

fclose(fptr);

printf("Student data loaded successfully from file.\n");

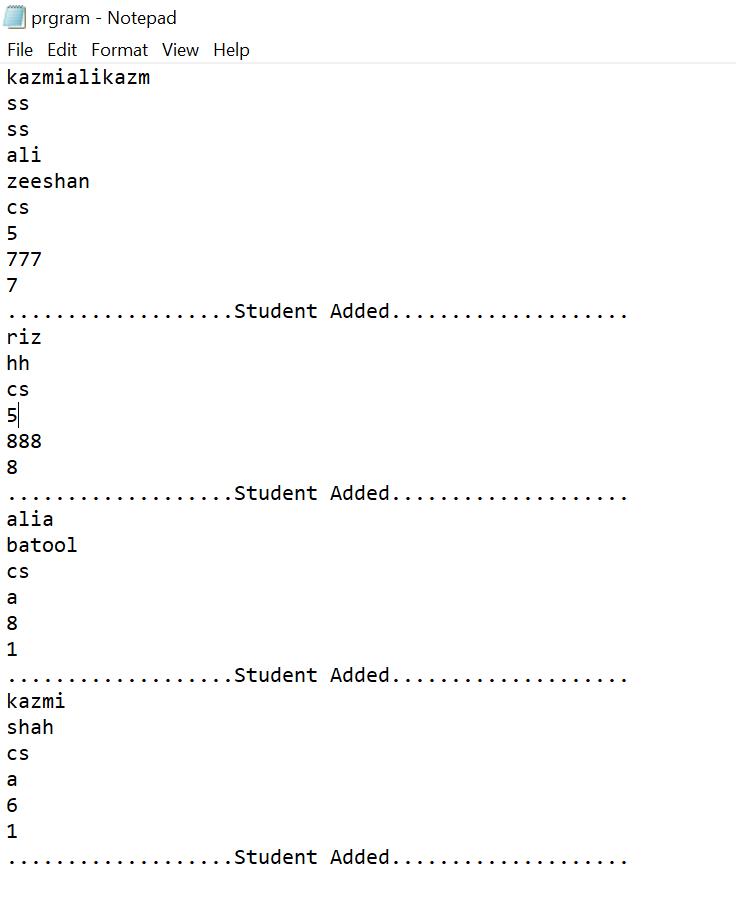
}





**6. Benefits of File Handling**

1. **Data Persistence**: The file ensures that student data is not lost when the program terminates.
2. **Scalability**: Writing to a file allows the program to handle more data than what can be stored in memory alone.
3. **Portability**: The data can be shared and used with other systems.



**7. Program Strengths**

* **Simplicity**: The menu-driven approach ensures easy navigation.
* **Scalability**: Handles up to 100 records, suitable for small institutions.
* **Modularity**: Functions are well-separated, making the program maintainable and extensible.
* **Robustness**: Adequate error handling ensures reliable performance.

**8. Limitations and Future Improvements**

1. **Limitations**:
   * Fixed maximum limit of 100 students.
   * No file storage for data persistence (data is lost after exiting).
   * No advanced input validation for string fields (e.g., special characters).
2. **Future Improvements**:
   * Implement file I/O to save and load student records.
   * Add more fields, such as date of birth or contact information.
   * Enable sorting and filtering of student records.
   * Create a graphical user interface (GUI) for better usability.

### ****8. Conclusion****

The **Student Information System** is a robust, foundational program for managing student records. By combining the power of structures, modular functions, and file handling, it provides a functional and user-friendly solution. While the system is effective in its current form, implementing the recommended improvements would greatly enhance its usability, scalability, and reliability.