**TASK 4 -**

# Test Case Table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case Description** | |  | | --- | | **Steps to Execute** | | |  | | --- | | **Steps to Execute** | | |  | | --- | | **Steps to Execute** | | **Pass/Fail** |
| |  | | --- | | Check available seats before registration |  |  | | --- | |  | | |  | | --- | | Press 1 to check available seats |  |  | | --- | |  | | Displays the number of available seats | Displays the correct number of available seats | Pass |
| |  | | --- | | Register a new student |  |  | | --- | |  | | |  | | --- | | Press 2, enter student ID and name |  |  | | --- | |  | | |  | | --- | | Student registered successfully, and available seats decrease by 1 |  |  | | --- | |  | | |  | | --- | | Student registered successfully, and available seats decrease by 1 |  |  | | --- | |  | | |  | | --- | | Pass |  |  | | --- | |  | |
| |  | | --- | | Delete a registered student |  |  | | --- | |  | | |  | | --- | | Press 3, enter student ID to delete |  |  | | --- | |  | | |  | | --- | | Student deleted successfully, and available seats increase by 1 |  |  | | --- | |  | | |  | | --- | | Student deleted successfully, and available seats increase by 1 |  |  | | --- | |  | | |  | | --- | | Pass |  |  | | --- | |  | |
| |  | | --- | | Find an existing student |  |  | | --- | |  | | |  | | --- | | Press 4, enter student ID to find |  |  | | --- | |  | | |  | | --- | | Displays student details |  |  | | --- | |  | | |  | | --- | | Displays student details |  |  | | --- | |  | | |  | | --- | | Pass |  |  | | --- | |  | |
| |  | | --- | | Store student details to file |  |  | | --- | |  | | |  | | --- | | Press 5 |  |  | | --- | |  | | |  | | --- | | Details stored in students.txt file |  |  | | --- | |  | | |  | | --- | | Details stored in students.txt file |  |  | | --- | |  | | |  | | --- | | Pass |  |  | | --- | |  | |
| |  | | --- | | Load student details from file |  |  | | --- | |  | | |  | | --- | | Press 6 |  |  | | --- | |  | | |  | | --- | | Loads details from students.txt file and displays them |  |  | | --- | |  | | |  | | --- | | Loads details from students.txt file and displays them |  |  | | --- | |  | | |  | | --- | | Pass |  |  | | --- | |  | |
| |  | | --- | | View sorted student list |  |  | | --- | |  | | |  | | --- | | Press 7 |  |  | | --- | |  | | |  | | --- | | Displays list of students sorted by name |  |  | | --- | |  | | |  | | --- | | Displays list of students sorted by name |  |  | | --- | |  | | |  | | --- | | Pass |  |  | | --- | |  | |
| |  | | --- | | Add student name and module marks |  |  | | --- | |  | | |  | | --- | | Press 8, choose sub-option 1 to add name and sub-option 2 to add marks |  |  | | --- | |  | | |  | | --- | | Adds student name and marks successfully |  |  | | --- | |  | | |  | | --- | | Adds student name and marks successfully |  |  | | --- | |  | | |  | | --- | | Pass |  |  | | --- | |  | |
| |  | | --- | | Generate system summary report |  |  | | --- | |  | | |  | | --- | | Press 9, choose sub-option C |  |  | | --- | |  | | |  | | --- | | Displays system summary report |  |  | | --- | |  | | |  | | --- | | Displays system summary report |  |  | | --- | |  | | Pass |
| |  | | --- | | Generate complete report |  |  | | --- | |  | | Press 9, choose sub-option D | |  | | --- | | Displays complete report sorted by average marks |  |  | | --- | |  | | |  | | --- | | Displays complete report sorted by average marks |  |  | | --- | |  | | Pass |
| Exit | Press 0 | Exiting | Exiting | Pass |

# Discussion of Test Case Selection

## Core Functionalities:

* **Registration and Deletion:** The system was tested for both valid and invalid student registration and deletion scenarios to make sure all edge cases are handled appropriately.
* **ID Validation:** During registration, the accuracy and uniqueness of student IDs were verified.
* **Finding Students:** Verified that the system can correctly locate existing students and handle cases where the student ID does not exist.

### Data Persistence:

* **Storing and Loading Data:** Confirmed that data is correctly saved to and loaded from a file, ensuring the program maintains student information across sessions.

### Results Management:

* **Updating Results:** Ensured that the system accurately updates and stores the results for each student.
* **Grade Calculation:** Verified the calculation of grades based on marks input, testing both passing and failing scenarios.

### Reporting:

* **Generating Reports:** Every report type has been tested to guarantee that accurate and insightful data is produced. Reports for students who pass every module, fail one or more, and have the best or lowest grades in a particular module are included in this.

### User Interaction:

* **Menu Navigation:** Ensured the menu options function as expected and return to the main menu when appropriate.
* **Input Handling:** Tested how the system handles both valid and invalid user inputs across different functionalities.

# Comparison: Array-Based vs. Class-Based Solution

### Array-Based Solutions

#### Pros:

* **Simplicity:** The logic is straightforward and easy to follow, especially for smaller datasets.
* **Less Overhead:** Arrays are generally less complex, which can lead to better performance in terms of memory and processing for small-scale applications.

#### Cons:

* **Scalability:** Arrays are not dynamically resizable, making it difficult to handle varying numbers of students or modules without additional logic for resizing or managing array bounds.
* **Readability:** As the program grows, managing multiple parallel arrays can become cumbersome and harder to understand.
* **Modification:** Adding new features or modifying the data structure would require significant changes across the program, leading to more error-prone code.

### Class-Based Solution

#### Pros:

* **Modularity:** The use of classes allows for a clear separation of concerns, making the code more organized and easier to maintain.
* **Scalability:** Classes and objects can be easily extended to add new attributes or methods without significantly altering existing code.
* **Readability:** With a well-defined structure, the class-based approach is generally more readable, especially for complex data relationships and operations.
* **Reusability:** Encapsulation and inheritance provide a foundation for code reuse and extension.

#### Cons:

* **Initial Complexity:** The class-based solution can be more complex to implement initially, especially for developers who are less familiar with object-oriented concepts.
* **Overhead:** There may be slight performance overhead due to the additional abstraction layers, though this is usually negligible for most applications.