

Student Management System Project Documentation

Course: Data Structures Project Name: Student Management System

Submitted by	
Name: Omar Elsayed Abd Elaziz	ID:224101541
Name: Sara Mohamed Awad	ID:224101527
Name: Yousef Mouinr Khader	ID: 224101574

1. Project Overview

Description

The Student Management System is a console-based application developed in C++ designed to manage student records within a university efficiently. The system replaces static array-based storage with Linked Lists, allowing for dynamic memory management where the system allocates memory only as needed.

Purpose

The primary goal of this project is to apply fundamental Data Structures concepts, specifically Singly Linked Lists, Pointers, and Dynamic Memory Allocation. It aims to provide a robust interface for administrators to perform CRUD (Create, Read, Update, Delete) operations on student data while ensuring data integrity through validation and file persistence.

Main Functions

Dynamic Storage: Uses a linked list to store an unlimited number of students (limited only by system memory).

Data Management: Allows adding, searching, updating, and deleting student records.

Data Persistence: Saves data to a CSV file (Excel compatible) and loads it upon program startup.

Advanced Sorting: Sorts students by Name or GPA in Ascending or Descending order.

Filtering: Filters and displays students based on a minimum GPA threshold.

Input Validation: Robust handling of user inputs (IDs, Names, Integers, Floats) to prevent crashes.

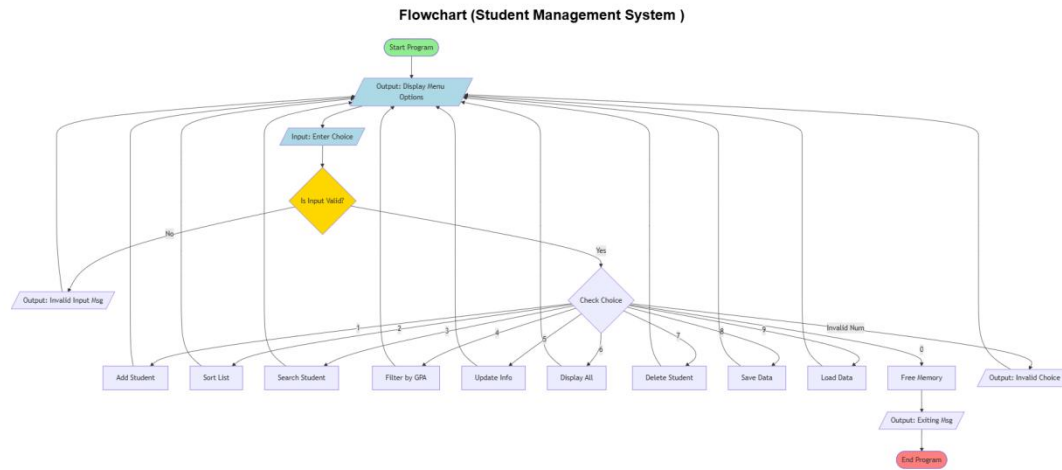
2. System Design

Main Modules and Functions

The system is built using a modular approach. The core structure is `Student_info`, which acts as the node for the linked list.

Function Name	Description
Valid_Int / Valid_Float	Helper functions that read input as strings and validate them to ensure they are numbers, preventing program crashes.
Valid_Name / Format_Name	Ensures names contain only letters and formats them (e.g., "ali ahmed" becomes "Ali Ahmed").
Add_Student	Creates a new node, validates the ID for uniqueness, and appends it to the end of the linked list.
Display_Student	Traverses the linked list from head to nullptr and prints details for every student, including a total count.
Search_Student	Performs a linear search to find a student by their unique ID or Name.
Update_Student	Locates a student by ID and allows the user to modify their Name, Age, GPA, or Grades.
Delete_Student	Removes a specific node from the linked list and frees the allocated memory.
Sort_Student	Implements the Bubble Sort algorithm to rearrange nodes based on user criteria (Name/GPA).
Filter_Students	Traverses the list and displays only students who meet a specific GPA threshold.
Save_Data / Load_Data	Handles File I/O operations to save the list to <code>Student.csv</code> and reload it.

2. Program Flowchart:



Start -> Display Menu (Add, Sort, Search, Filter, Update, Display, Delete, Save, Load, Exit).

User Input -> Validate Input.

Decision:

If Add: Call Add_Student -> Return to Menu.

If Display: Call Display_Student -> Return to Menu.

If Sort: Ask Criteria -> Call Sort_Student -> Return to Menu.

If Filter: Ask GPA -> Call Filter_Students -> Return to Menu.

If Save/Load: Perform File Operation -> Return to Menu.

If Exit (0): Save Data -> Free Memory -> End.

3. Additional Features Implemented

To enhance the system's capability, the following advanced features were implemented:

A. Save and Load Feature (Data Persistence):

Instead of losing data when the program closes, the system writes the linked list data to a Student.csv file.

Saving: Iterates through the list and writes data in Comma-Separated Values format.

Loading: Reads the file line by line, parses the CSV format, creates new nodes, and reconstructs the linked list automatically.

B. Unique Sorting Options:

The system goes beyond basic sorting by offering dynamic choices:

Criteria: Users can sort by Name or GPA.

Order: Users can choose Ascending or Descending order.

Implementation: Uses a Bubble Sort algorithm that swaps data between nodes based on the selected condition.

C. Filtering Functionality:

A feature that allows the user to view a specific subset of students.

The user inputs a minimum GPA (e.g., 3.0).

The system traverses the list and prints only the students whose GPA is equal to or higher than the input.

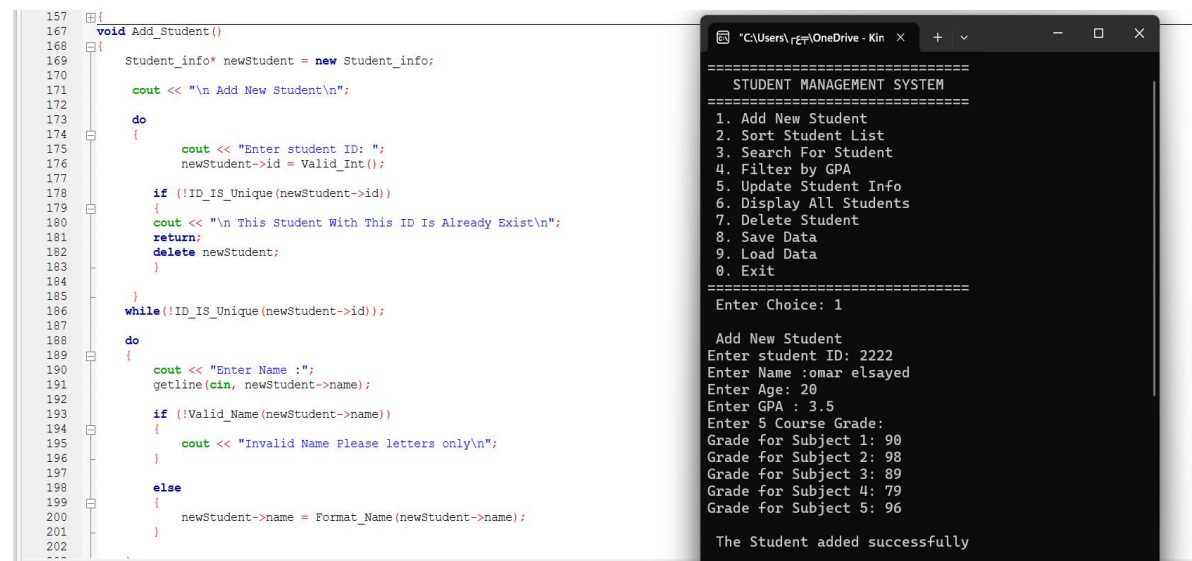
4. Testing and Validation:

Test Case 1: Adding and displaying students

Scenario: Adding a new student with valid data and formatting the name automatically.

Input: ID: 2222, Name: Omar Elsayed, Age: 20, GPA: 3.5.

Expected Output: Name converted to "Omar Elsayed", student added successfully, and displayed in the list.



```
157 void Add_Student()
158 {
159     Student_info* newStudent = new Student_info;
160     cout << "\n Add New Student\n";
161     do
162     {
163         cout << "Enter student ID: ";
164         newStudent->id = Valid_Int();
165         if (!ID_Is_Unique(newStudent->id))
166         {
167             cout << "\n This Student With This ID Is Already Exist\n";
168             return;
169             delete newStudent;
170         }
171     } while (!ID_Is_Unique(newStudent->id));
172     do
173     {
174         cout << "Enter Name : ";
175         getline(cin, newStudent->name);
176         if (!Valid_Name(newStudent->name))
177         {
178             cout << "Invalid Name Please letters only\n";
179         }
180         else
181         {
182             newStudent->name = Format_Name(newStudent->name);
183         }
184     } while (!Valid_Name(newStudent->name));
185     do
186     {
187         cout << "Enter Age: ";
188         newStudent->age = Valid_Int();
189         if (newStudent->age < 17 || newStudent->age > 50)
190         {
191             cout << "Age Cannot Be More Than 50 Or less than 17 \n";
192         }
193     } while (newStudent->age < 17 || newStudent->age > 50);
194     do
195     {
196         cout << "Enter GPA : ";
197         newStudent->gpa = Valid_Float();
198         if (newStudent->gpa > 4.0) {
199             cout << " GPA cannot be more than 4.0. Try again.\n";
200         }
201     } while (newStudent->gpa > 4.0);
202     cout << "Enter 5 Course Grade:\n";
203     for (int i = 0; i < 5; i++)
204     {
205         do {
206             cout << "Grade for Subject " << (i + 1) << ": ";
207             float g = Valid_Float();
208             if (g > 100) {
209                 cout << " Grade cannot be more than 100. Try again.\n";
210             }
211         } while (g > 100);
212         newStudent->grades[i] = g;
213     }
214     Save_Data();
215     Load_Data();
216     Display_All_Students();
217 }
```

```
=====
STUDENT MANAGEMENT SYSTEM
=====
1. Add New Student
2. Sort Student List
3. Search For Student
4. Filter by GPA
5. Update Student Info
6. Display All Students
7. Delete Student
8. Save Data
9. Load Data
0. Exit
=====
Enter Choice: 1

Add New Student
Enter student ID: 2222
Enter Name :omar elsayed
Enter Age: 20
Enter GPA : 3.5
Enter 5 Course Grade:
Grade for Subject 1: 90
Grade for Subject 2: 98
Grade for Subject 3: 89
Grade for Subject 4: 79
Grade for Subject 5: 96

The Student added successfully
```

```
202     }
203     while (!Valid_Name(newStudent->name));
204     do
205     {
206         cout << "Enter Age: ";
207         newStudent->age = Valid_Int();
208         if (newStudent->age < 17 || newStudent->age > 50)
209         {
210             cout << "Age Cannot Be More Than 50 Or less than 17 \n";
211         }
212     } while (newStudent->age < 17 || newStudent->age > 50);
213     do
214     {
215         cout << "Enter GPA : ";
216         newStudent->gpa = Valid_Float();
217         if (newStudent->gpa > 4.0) {
218             cout << " GPA cannot be more than 4.0. Try again.\n";
219         }
220     } while (newStudent->gpa > 4.0);
221     cout << "Enter 5 Course Grade:\n";
222     for (int i = 0; i < 5; i++)
223     {
224         do {
225             cout << "Grade for Subject " << (i + 1) << ": ";
226             float g = Valid_Float();
227             if (g > 100) {
228                 cout << " Grade cannot be more than 100. Try again.\n";
229             }
230         } while (g > 100);
231         newStudent->grades[i] = g;
232     }
233     Save_Data();
234     Load_Data();
235     Display_All_Students();
236 }
```

```

226     }
227 }
228 while (newStudent->gpa > 4.0);
229
230 cout << "Enter 5 Course Grade:\n";
231 for (int i = 0; i < 5; i++)
232 {
233     do {
234         cout << "Grade for Subject " << (i + 1) << ": ";
235         float g = Valid_Float();
236
237         if (g > 100) {
238             cout << " Grade cannot be more than 100. Try again.\n";
239         } else {
240             newStudent->grades[i].grades = g;
241             break;
242         }
243     } while (true);
244 }
245
246 newStudent->next_pointer = nullptr;
247
248 if (head == nullptr)
249 {
250     head = newStudent;
251 }
252 else
253 {
254     Student_info* temp = head;
255     while (temp->next_pointer != nullptr)
256     {
257         temp = temp->next_pointer;
258     }
259     temp->next_pointer = newStudent;
260 }
261 cout << "\n The Student added successfully\n";
262 }

```

```

157 void Add_Student()
158 {
263 void Display_Student()
264 {
265     if (head == nullptr)
266     {
267         cout << "No Student in the list\n";
268         return;
269     }
270
271     Student_info* temp = head;
272     int studentCount = 0;
273
274     cout << "====\n All Student List\n====\n";
275
276     while (temp != nullptr)
277     {
278         studentCount++;
279
280         cout << "ID: " << temp->id << endl;
281         cout << "Name: " << temp->name << endl;
282         cout << "Age: " << temp->age << endl;
283         cout << "GPA: " << temp->gpa << endl;
284
285         cout << "Grades:\n";
286
287         for (int i = 0; i < 5; i++)
288         {
289             cout << "Subject " << i + 1 << ": "
290                 << temp->grades[i].grades << endl;
291         }
292
293         cout << "-----\n";
294         temp = temp->next_pointer;
295     }
296
297     cout << "Total Number of Students: " << studentCount << endl;
298     cout << "=====\n";
299 }
300

```

```

=====
1. Add New Student
2. Sort Student List
3. Search For Student
4. Filter by GPA
5. Update Student Info
6. Display All Students
7. Delete Student
8. Save Data
9. Load Data
0. Exit
=====
Enter Choice: 6
=====
All Student List
=====
ID: 2222
Name: Omar Elsayed
Age: 20
GPA: 3.5
Grades:
Subject 1: 90
Subject 2: 98
Subject 3: 89
Subject 4: 79
Subject 5: 96
=====
Total Number of Students: 1
=====

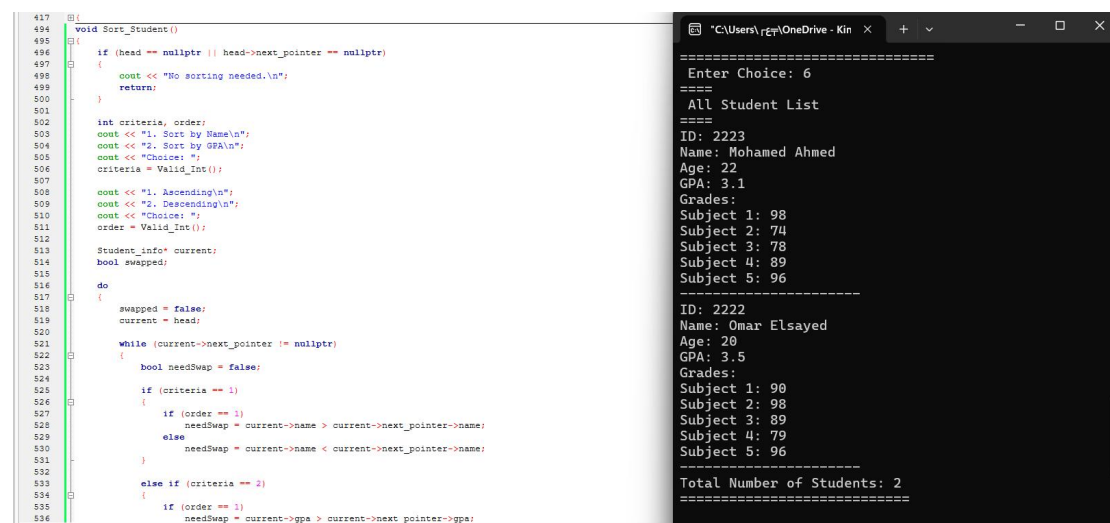
```

Test Case 2: Sorting and Filtering

Scenario: Sorting the list by GPA (Descending) and Filtering students with GPA > 3.0.

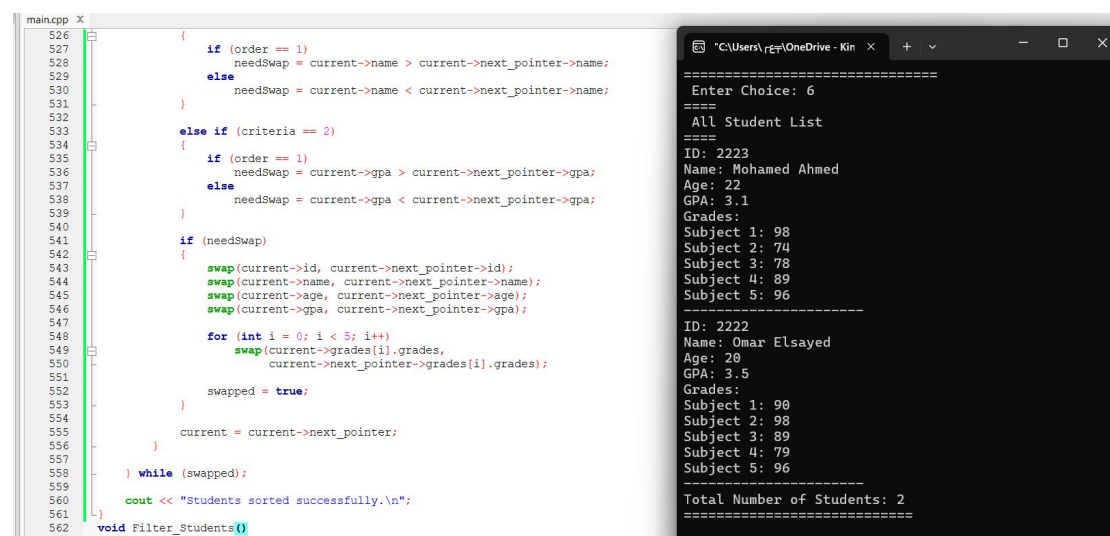
Action: Choose Option 2 (Sort -> GPA -> Descending), then Option 4 (Filter -> 3.0).

Expected Output: The list is rearranged from highest GPA to lowest, and the filter shows only top students.



```
417 void Sort_Student()
418 {
419     if (head == nullptr || head->next_pointer == nullptr)
420     {
421         cout << "No sorting needed.\n";
422         return;
423     }
424
425     int criteria, order;
426     cout << "1. Sort by Name\n";
427     cout << "2. Sort by GPA\n";
428     cout << "Choice: ";
429     criteria = Valid_Int();
430
431     cout << "1. Ascending\n";
432     cout << "2. Descending\n";
433     cout << "Choice: ";
434     order = Valid_Int();
435
436     Student_info* current;
437     bool swapped;
438
439     do
440     {
441         swapped = false;
442         current = head;
443
444         while (current->next_pointer != nullptr)
445         {
446             bool needSwap = false;
447
448             if (criteria == 1)
449             {
450                 if (order == 1)
451                     needSwap = current->name > current->next_pointer->name;
452                 else
453                     needSwap = current->name < current->next_pointer->name;
454             }
455             else if (criteria == 2)
456             {
457                 if (order == 1)
458                     needSwap = current->gpa > current->next_pointer->gpa;
459             }
460
461             if (needSwap)
462             {
463                 swap(current->id, current->next_pointer->id);
464                 swap(current->name, current->next_pointer->name);
465                 swap(current->age, current->next_pointer->age);
466                 swap(current->gpa, current->next_pointer->gpa);
467
468                 for (int i = 0; i < 5; i++)
469                     swap(current->grades[i].grades, current->next_pointer->grades[i].grades);
470
471                 swapped = true;
472             }
473
474             current = current->next_pointer;
475         }
476     } while (swapped);
477
478     cout << "Students sorted successfully.\n";
479 }
480
481 void Filter_Students()
```

```
Enter Choice: 6
=====
All Student List
=====
ID: 2223
Name: Mohamed Ahmed
Age: 22
GPA: 3.1
Grades:
Subject 1: 98
Subject 2: 74
Subject 3: 78
Subject 4: 89
Subject 5: 96
=====
ID: 2222
Name: Omar Elsayed
Age: 20
GPA: 3.5
Grades:
Subject 1: 90
Subject 2: 98
Subject 3: 89
Subject 4: 79
Subject 5: 96
=====
Total Number of Students: 2
=====
```



```
526 {
527     if (order == 1)
528         needSwap = current->name > current->next_pointer->name;
529     else
530         needSwap = current->name < current->next_pointer->name;
531 }
532
533 else if (criteria == 2)
534 {
535     if (order == 1)
536         needSwap = current->gpa > current->next_pointer->gpa;
537     else
538         needSwap = current->gpa < current->next_pointer->gpa;
539 }
540
541 if (needSwap)
542 {
543     swap(current->id, current->next_pointer->id);
544     swap(current->name, current->next_pointer->name);
545     swap(current->age, current->next_pointer->age);
546     swap(current->gpa, current->next_pointer->gpa);
547
548     for (int i = 0; i < 5; i++)
549         swap(current->grades[i].grades, current->next_pointer->grades[i].grades);
550
551     swapped = true;
552 }
553
554 current = current->next_pointer;
555 } while (swapped);
556
557 cout << "Students sorted successfully.\n";
558 }
559
560 void Filter_Students()
```

```
Enter Choice: 6
=====
All Student List
=====
ID: 2223
Name: Mohamed Ahmed
Age: 22
GPA: 3.1
Grades:
Subject 1: 98
Subject 2: 74
Subject 3: 78
Subject 4: 89
Subject 5: 96
=====
ID: 2222
Name: Omar Elsayed
Age: 20
GPA: 3.5
Grades:
Subject 1: 90
Subject 2: 98
Subject 3: 89
Subject 4: 79
Subject 5: 96
=====
Total Number of Students: 2
=====
```

```

562 void Filter_Students()
563 {
564     if (head == nullptr) {
565         cout << "List is empty.\n";
566         return;
567     }
568     float minGPA;
569     cout << "==== Filter Students =====\n";
570     cout << "Enter Minimum GPA to Display: ";
571     minGPA = Valid_Float();
572     Student_info* temp = head;
573     bool found = false;
574     while (temp != nullptr)
575     {
576         if (temp->gpa >= minGPA)
577         {
578             cout << "ID: " << temp->id
579                 << " | Name: " << temp->name
580                 << " | GPA: " << temp->gpa << endl;
581             found = true;
582         }
583         temp = temp->next_pointer;
584     }
585     if (!found) cout << "No students found with GPA higher than " << minGPA << "\n";
586     cout << "-----\n";
587 }
588
589
590
591
592

```

```

=====
Enter Choice: 4
==== Filter Students =====Enter Minimum GPA to Display: 3.0

--- Students with GPA >= 3 ---
ID: 2223 | Name: Mohamed Ahmed | GPA: 3.1
ID: 2222 | Name: Omar Elsayed | GPA: 3.5
-----

STUDENT MANAGEMENT SYSTEM
=====
1. Add New Student
2. Sort Student List
3. Search For Student
4. Filter by GPA
5. Update Student Info
6. Display All Students
7. Delete Student
8. Save Data
9. Load Data
0. Exit
=====
Enter Choice: |

```

Test Case 3: Error Handling (Validation):

Scenario: Attempting to enter invalid data (Duplicate ID and String in Integer field).

Action 1: Try to add a student with an ID that already exists.

Action 2: Try to enter text (e.g., "twenty") in the Age or ID field.

Expected Output:

"This Student With This ID Is Already Exist".

"Invalid input. Please enter a number only."

```

int ID_Is_Unique(int id)
{
    Student_info* temp = head;
    while (temp != nullptr) {
        if (temp->id == id) {
            return false;
        }
        temp = temp->next_pointer;
    }
    return true;
}

```

```

=====
Enter Choice: 1
Add New Student
Enter student ID: 2222

This Student With This ID Is Already Exist
=====
STUDENT MANAGEMENT SYSTEM
=====
1. Add New Student
2. Sort Student List
3. Search For Student
4. Filter by GPA
5. Update Student Info
6. Display All Students
7. Delete Student
8. Save Data
9. Load Data
0. Exit
=====
Enter Choice: fgdgfdg
Invalid input. Please Enter a number only (no letters):

```

```

74 int Valid_Int()
75 {
76     string line;
77     while (true)
78     {
79         getline(cin, line);
80         string cleanLine = "";
81         for (char c : line) {
82             if (!isspace(c)) {
83                 cleanLine += c;
84             }
85         }
86         if (cleanLine.empty()) {
87             cout << "Invalid input. Please Enter a number only: ";
88             continue;
89         }
90         bool valid = true;
91         for (char c : cleanLine) {
92             if (!isdigit(c)) {
93                 valid = false;
94                 break;
95             }
96         }
97         if (!valid) {
98             cout << "Invalid input. Please Enter a number only (no letters): ";
99             continue;
100         }
101         return stoi(cleanLine);
102     }
103 }

```

```

This Student With This ID Is Already Exist

STUDENT MANAGEMENT SYSTEM
=====
1. Add New Student
2. Sort Student List
3. Search For Student
4. Filter by GPA
5. Update Student Info
6. Display All Students
7. Delete Student
8. Save Data
9. Load Data
0. Exit
=====
Enter Choice: fgdgfdg
Invalid input. Please Enter a number only (no letters):

```

```

class Course_Grades
{
class Student_info
{
Student info* head = nullptr;
bool Valid_Name(string name)
{
if (name.length() == 0)
return false;
for (int i = 0; i < name.length(); i++)
{
if (!isalpha(name[i]) )
{
return false;
}
else if (!isspace(name[i]))
{
return true ;
}
else if (!isdigit(name[i]))
{
return false;
}
}
return true;
}
}
}

```

```

=====
STUDENT MANAGEMENT SYSTEM
=====
1. Add New Student
2. Sort Student List
3. Search For Student
4. Filter by GPA
5. Update Student Info
6. Display All Students
7. Delete Student
8. Save Data
9. Load Data
0. Exit
=====
Enter Choice: 1

Add New Student
Enter student ID: 22312
Enter Name :32323
Invalid Name Please letters only
Enter Name :mohamed elsayed
Enter Age:

```

```

do
{
cout << "Enter GPA : ";
newStudent->gpa = Valid_Float();
if (newStudent->gpa > 4.0) {
cout << " GPA cannot be more than 4.0. Try again.\n";
}
}
while (newStudent->gpa > 4.0);

```

```

Add New Student
Enter student ID: 22312
Enter Name :32323
Invalid Name Please letters only
Enter Name :mohamed elsayed
Enter Age: 22
Enter GPA : 32
GPA cannot be more than 4.0. Try again.
Enter GPA : 2.3
Enter 5 Course Grade:
Grade for Subject 1:

```

```

while (!Valid_Name(newStudent->name));
do
{
cout << "Enter Age: ";
newStudent->age = Valid_Int();
if (newStudent->age < 17 || newStudent->age > 50)
{
cout << "Age Cannot Be More Than 50 Or less than 17 \n";
}
}
while (newStudent->age < 17 || newStudent->age > 50);

```

```

=====
Enter Choice: 1

Add New Student
Enter student ID: 22312
Enter Name :32323
Invalid Name Please letters only
Enter Name :mohamed elsayed
Enter Age: 22
Enter GPA : 32
GPA cannot be more than 4.0. Try again.
Enter GPA : 2.3
Enter 5 Course Grade:
Grade for Subject 1: |

```

Test Case 4: File Persistence (Excel)

Scenario: Saving data and opening the file in Excel.

Action: Choose Option 8 (Save Data). Open Student.csv on the computer.

Expected Output: Data is organized neatly in columns (ID, Name, Age, GPA, Grades).

A	B	C	D	E	F	G	H	I	J
ID	Name	Age	GPA	Subject 1	Subject 2	Subject 3	Subject 4	Subject 5	
2222	Omar Elsayed	20	3.5	90	98	89	79	96	
2223	Mohamed Ahmed	22	3.1	98	74	78	89	96	
2214	Mohamed Omar	23	3.4	100	70	98	97	89	