

# 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.

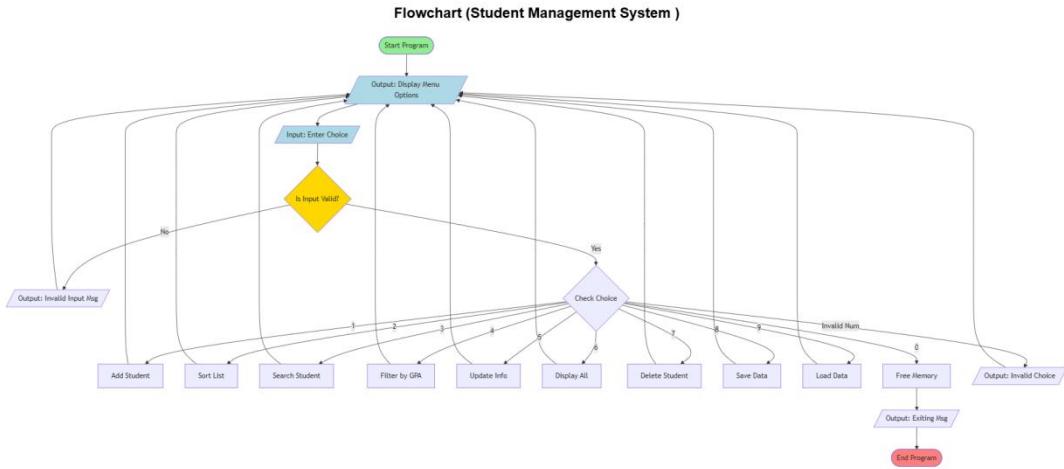
## 2. System Design

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

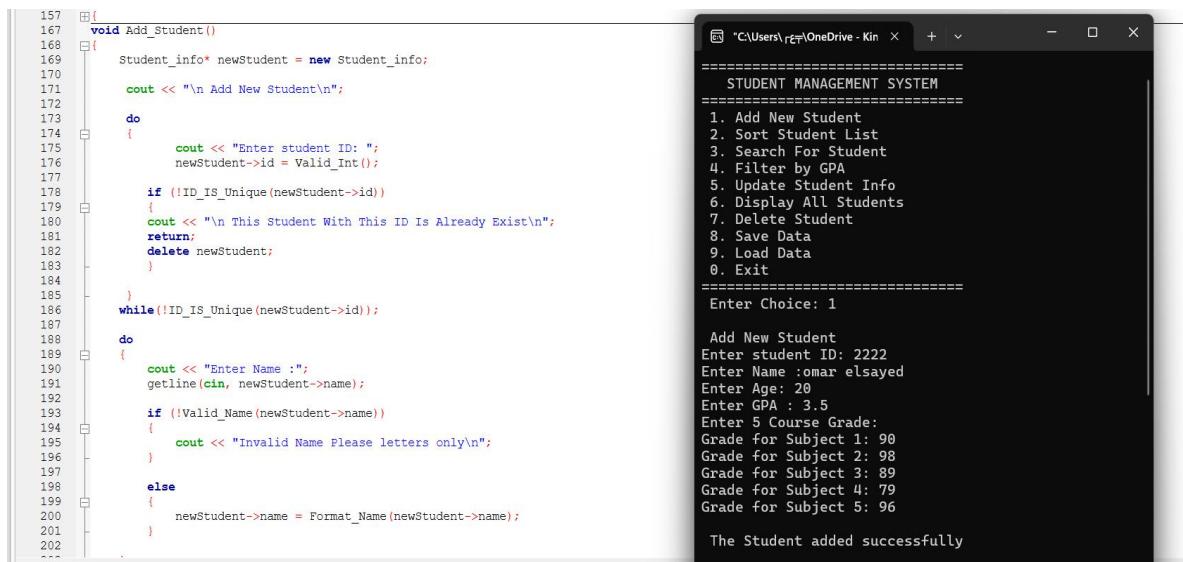
## 4. Testing and Validation:

Test Case 1: Adding and displaying students

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

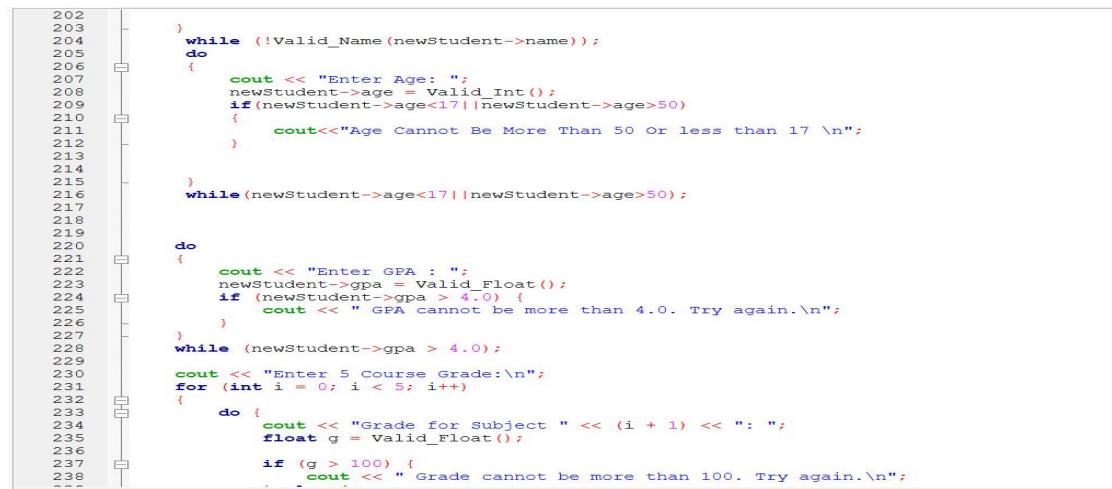
Input: ID: 2222, Name: Omar Elsayed, Age: 20, GPA: 3.5.

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
161     cout << "\n Add New Student\n";
162
163     do
164     {
165         cout << "Enter student ID: ";
166         newStudent->id = Valid_Int();
167
168         if (!ID_IS_Unique(newStudent->id))
169         {
170             cout << "\n This Student With This ID Is Already Exist\n";
171             return;
172         }
173         delete newStudent;
174     }
175
176     while (!ID_IS_Unique(newStudent->id));
177
178     do
179     {
180         cout << "Enter Name :";
181         getline(cin, newStudent->name);
182
183         if (!Valid_Name(newStudent->name))
184         {
185             cout << "Invalid Name Please letters only\n";
186         }
187         else
188         {
189             newStudent->name = Format_Name(newStudent->name);
190         }
191     }
192
193     cout << "Enter Age :";
194     newStudent->age = Valid_Int();
195
196     if (newStudent->age < 17 || newStudent->age > 50)
197     {
198         cout << "Age Cannot Be More Than 50 Or less than 17 \n";
199     }
200
201     cout << "Enter GPA : ";
202     newStudent->gpa = Valid_Float();
203
204     if (newStudent->gpa > 4.0)
205     {
206         cout << "GPA cannot be more than 4.0. Try again.\n";
207     }
208
209     cout << "Enter 5 Course Grade:\n";
210     for (int i = 0; i < 5; i++)
211     {
212         cout << "Grade for Subject " << (i + 1) << ":" ;
213         float g = Valid_Float();
214
215         if (g > 100)
216         {
217             cout << " Grade cannot be more than 100. Try again.\n";
218         }
219     }
220
221     cout << "The Student added successfully";
222 }
```

```
*C:\Users\ezz\OneDrive - Kin * + - □ ×
=====
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
204     }
205     while (!Valid_Name(newStudent->name));
206     do
207     {
208         cout << "Enter Age : ";
209         newStudent->age = Valid_Int();
210         if (newStudent->age < 17 || newStudent->age > 50)
211         {
212             cout << "Age Cannot Be More Than 50 Or less than 17 \n";
213         }
214
215     }
216     while (newStudent->age < 17 || newStudent->age > 50);
217
218
219     do
220     {
221         cout << "Enter GPA : ";
222         newStudent->gpa = Valid_Float();
223         if (newStudent->gpa > 4.0)
224         {
225             cout << "GPA cannot be more than 4.0. Try again.\n";
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         {
235             cout << "Grade for Subject " << (i + 1) << ":" ;
236             float g = Valid_Float();
237
238             if (g > 100)
239             {
240                 cout << " Grade cannot be more than 100. Try again.\n";
241             }
242         }
243     }
244 }
```

```
226
227
228     }
229
230     while (newStudent->gpa > 4.0);
231
232     cout << "Enter 5 Course Grade:\n";
233     for (int i = 0; i < 5; i++)
234     {
235         do {
236             cout << "Grade for Subject " << (i + 1) << ": ";
237             float g = Valid_Float();
238
239             if (g > 100) {
240                 cout << "Grade cannot be more than 100. Try again.\n";
241             } else {
242                 newStudent->grades[i].grades = g;
243                 break;
244             }
245         } while (true);
246     }
247
248     newStudent->next_pointer = nullptr;
249
250     if (head == nullptr)
251     {
252         head = newStudent;
253     }
254     else
255     {
256         Student_info* temp = head;
257         while (temp->next_pointer != nullptr)
258         {
259             temp = temp->next_pointer;
260         }
261         temp->next_pointer = newStudent;
262     }
263     cout << "\n The Student added successfully\n";
264 }
```

## 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.

The screenshot shows a code editor with the `Sort_Student()` function highlighted. The function takes a `Student_info** head` parameter and sorts the list based on user input for criteria and order. The execution output window shows the sorted student list and the total number of students.

```
417 void Sort_Student()
494 {
495     if (head == nullptr || head->next_pointer == nullptr)
496     {
497         cout << "No sorting needed.\n";
498         return;
499     }
500
501     int criteria, order;
502     cout << "1. Sort by Name\n";
503     cout << "2. Sort by GPA\n";
504     cout << "Choice: ";
505     criteria = Valid_Int();
506
507     cout << "1. Ascending\n";
508     cout << "2. Descending\n";
509     cout << "Choice: ";
510     order = Valid_Int();
511
512     Student_info* current;
513     bool swapped;
514
515     do
516     {
517         swapped = false;
518         current = head;
519
520         while (current->next_pointer != nullptr)
521         {
522             bool needSwap = false;
523
524             if (criteria == 1)
525             {
526                 if (order == 1)
527                     needSwap = current->name > current->next_pointer->name;
528                 else
529                     needSwap = current->name < current->next_pointer->name;
530             }
531
532             else if (criteria == 2)
533             {
534                 if (order == 1)
535                     needSwap = current->gpa > current->next_pointer->gpa;
536             }
537         }
538
539         if (needSwap)
540         {
541             swap(current->id, current->next_pointer->id);
542             swap(current->name, current->next_pointer->name);
543             swap(current->age, current->next_pointer->age);
544             swap(current->gpa, current->next_pointer->gpa);
545
546             for (int i = 0; i < 5; i++)
547                 swap(current->grades[i].grades,
548                      current->next_pointer->grades[i].grades);
549
550             swapped = true;
551         }
552
553         current = current->next_pointer;
554     }
555
556     } while (!swapped);
557
558     cout << "Students sorted successfully.\n";
559
560     void Filter_Students();
561 }
```

```
=====
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
=====
```

The screenshot shows a code editor with the `main()` function highlighted. The function calls `Sort_Student()` and `Filter_Students()`. The execution output window shows the sorted student list and the total number of students.

```
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         else if (criteria == 2)
533         {
534             if (order == 1)
535                 needSwap = current->gpa > current->next_pointer->gpa;
536             else
537                 needSwap = current->gpa < current->next_pointer->gpa;
538
539             if (needSwap)
540             {
541                 swap(current->id, current->next_pointer->id);
542                 swap(current->name, current->next_pointer->name);
543                 swap(current->age, current->next_pointer->age);
544                 swap(current->gpa, current->next_pointer->gpa);
545
546                 for (int i = 0; i < 5; i++)
547                     swap(current->grades[i].grades,
548                          current->next_pointer->grades[i].grades);
549
550                 swapped = true;
551             }
552
553             current = current->next_pointer;
554         }
555
556     } while (!swapped);
557
558     cout << "Students sorted successfully.\n";
559
560     void Filter_Students();
561 }
```

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

```

## 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;
}

```

```

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 (!isalpha(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    }
}

```

```

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	