PROJECT

Student Name: Yuraj Singh **UID: 22BCA10824**
Branch: BCA **Section/Group: 3-B**
Semester: 5th **Date of Performance: 26/10/2024**
Subject Name: Web Development **Subject Code: 22CAH-301**

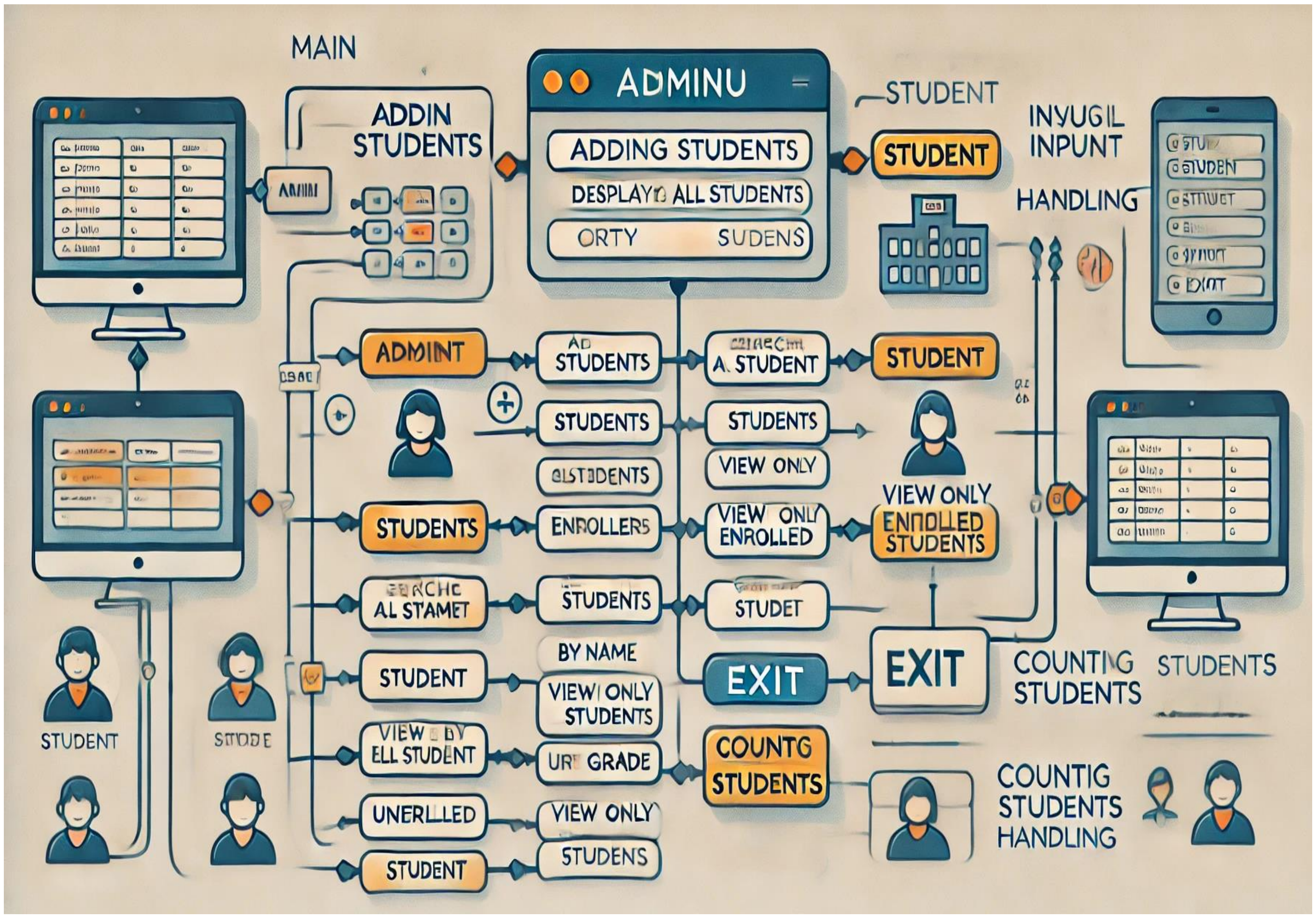
1. **Aim/Overview of the project.**

   The **School Management System** project is to develop a structured and efficient system for managing student records within a school. The project provides tools for administrators and students to interact with student data, supporting various functionalities such as adding, searching, and enrolling students.

2. **Task to be done.**

   - **Add Student**: Admins can add new students, specifying details like name, age, and grade.
   - **Search Student**: Students or admins can search for a student by name to view their details.
   - **Enrollment Management**: Students can be enrolled or unenrolled as per the requirements.
   - **Display Students**: Users can view a list of all students, sort by name or grade, and filter to show only enrolled students.
   - **Student Count**: Quickly view the current number of students in the school.

3. Algorithm/Flowchart.



4. Dataset.

Dataset Structure in Detail

1. **Name** (string):
   - **Description**: Stores the full name of the student.
   - **Data Type**: string
   - **Purpose**: Used as the primary identifier for students in the system, allowing users to search by name and view or manage individual student details.
   - **Example Values**: "Alice Johnson," "Bob Smith," "Carlos Mendes"
2. **Age** (int):
   - **Description**: Represents the age of the student.
   - **Data Type**: int
   - **Purpose**: Provides additional context for student records, useful in understanding the demographic range within the school.
   - **Example Values**: 15, 17, 18
3. **Grade** (string):
   - **Description**: Indicates the academic grade or class the student is currently in.
   - **Data Type**: string
   - **Purpose**: Allows sorting and categorization of students by their educational level, making it easier to manage or filter student information based on grade.
   - **Example Values**: "10th," "11th," "12th"

4. **IsEnrolled** (bool):
   - **Description**: A boolean flag that shows whether the student is currently enrolled in the school.
   - **Data Type**: bool
   - **Default Value**: false (students are not enrolled by default when added)
   - **Purpose**: Useful for filtering views so that users can see only actively enrolled students. This helps in situations where a student might be added to the system but is not yet confirmed as enrolled.
   - **Example Values**: true (enrolled), false (not enrolled)

How the Dataset is Used in the Application

- **Storage**: The School class maintains a list of Student objects (List<Student> students). This list functions as the main dataset, holding all student records.
- **Filtering and Searching**:
  - **Filter by Enrollment**: Users can filter the dataset to display only those students who are enrolled (IsEnrolled = true).
  - **Search by Name**: The system allows searching the dataset by the Name attribute to retrieve and display specific student details.
  - **Sorting by Grade or Name**: The dataset can be sorted based on Grade or Name to provide ordered views of student data, which helps in quickly accessing specific records.

Example of the Dataset in Practice

Consider the following example records for students in the system:

csharp

Copy code

```
// Adding student records to the dataset

school.AddStudent(new Student("Alice Johnson", 17, "11th"));
school.AddStudent(new Student("Bob Smith", 16, "10th"));

school.AddStudent(new Student("Carlos Mendes", 18, "12th"));
```

Each entry contains the Name, Age, Grade, and IsEnrolled attributes, forming a row in the dataset. This allows various operations:

- **Search**: Find "Alice Johnson" to retrieve her details.
- **Enroll**: Change IsEnrolled to true for students as they enroll.
- **View Enrolled Students Only**: Display students with IsEnrolled = true.
- **Sort by Name**: View students in alphabetical order by name.

5. Code for experiment/practical:

```
using System;

using System.Collections.Generic;

using System.Linq;


public class Student

{

    public string Name { get; set; }

    public int Age { get; set; }

    public string Grade { get; set; }
    public bool IsEnrolled { get; set; } = false;


    public Student(string name = "", int age = 0, string grade = "")
    {
        Name = name;
        Age = age;
        Grade = grade;
    }


    public override string ToString()
    {
        return $"Name: {Name}, Age: {Age}, Grade: {Grade}, Enrolled: {(IsEnrolled ?
"Yes" : "No")}";
    }
}


public class School
{
    private const int MaxStudents = 100;
    private readonly List<Student> students = new();


    public void AddStudent(Student student)
    {
        if (students.Count < MaxStudents)
        {
            students.Add(student);
            Console.WriteLine("Student added successfully!");
        }
        else
        {
            Console.WriteLine("School is at maximum capacity, cannot add more students.");
        }
    }

    public void DisplayStudents(bool onlyEnrolled = false, bool sorted = false, string sortBy = "name")
    {
        var studentList = onlyEnrolled ? students.Where(s => s.IsEnrolled).ToList() : new List<Student>(students);

        if (studentList.Count == 0)
        {
            Console.WriteLine("No students to display.");
            return;
        }

        if (sorted)
        {
            studentList = sortBy.ToLower() == "grade"

                ? studentList.OrderBy(s => s.Grade).ToList()

                : studentList.OrderBy(s => s.Name).ToList();

        }


        Console.WriteLine("Students in the School:");

        studentList.ForEach(student => Console.WriteLine(student));

    }


    public void SearchStudent(string name)

    {

        var student = students.Find(s => s.Name.Equals(name, 
StringComparison.OrdinalIgnoreCase));

        Console.WriteLine(student != null ? $"Student found:\n{student}" : "Student not 
found.");

    }


    public void EnrollStudent(string name)

    {

        var student = students.Find(s => s.Name.Equals(name, 
StringComparison.OrdinalIgnoreCase));

        if (student != null)

        {

            if (!student.IsEnrolled)
            {
                student.IsEnrolled = true;
                Console.WriteLine($"Student '{name}' has been enrolled.");
            }
            else
            {
                Console.WriteLine($"Student '{name}' is already enrolled.");
            }
        }
        else
        {
            Console.WriteLine("Student not found.");
        }
    }

    public void UnenrollStudent(string name)
    {
        var student = students.Find(s => s.Name.Equals(name,
StringComparison.OrdinalIgnoreCase));
        if (student != null)
        {
            if (student.IsEnrolled)
            {
                student.IsEnrolled = false;
                Console.WriteLine($"Student '{name}' has been unenrolled.");
            }
            else
            {
                Console.WriteLine($"Student '{name}' was not enrolled.");
            }
        }
        else
        {
            Console.WriteLine("Student not found.");
        }
    }

    public int GetStudentCount() => students.Count;
}

public class Program
{
    private static void AdminMenu(School school)
    {
        int choice;
        do
        {
```

```
Console.WriteLine("\nAdmin Menu");

Console.WriteLine("1. Add Student");

Console.WriteLine("2. Display Students");

Console.WriteLine("3. Display Enrolled Students Only");

Console.WriteLine("4. Display Students Sorted by Name");

Console.WriteLine("5. Display Students Sorted by Grade");

Console.WriteLine("6. Count Students in School");

Console.WriteLine("7. Exit");

Console.Write("Enter your choice: ");


if (int.TryParse(Console.ReadLine(), out choice))

{

    switch (choice)

    {

        case 1:

            AddStudentFlow(school);

            break;

        case 2:

            school.DisplayStudents();

            break;

        case 3:

            school.DisplayStudents(onlyEnrolled: true);

            break;
```

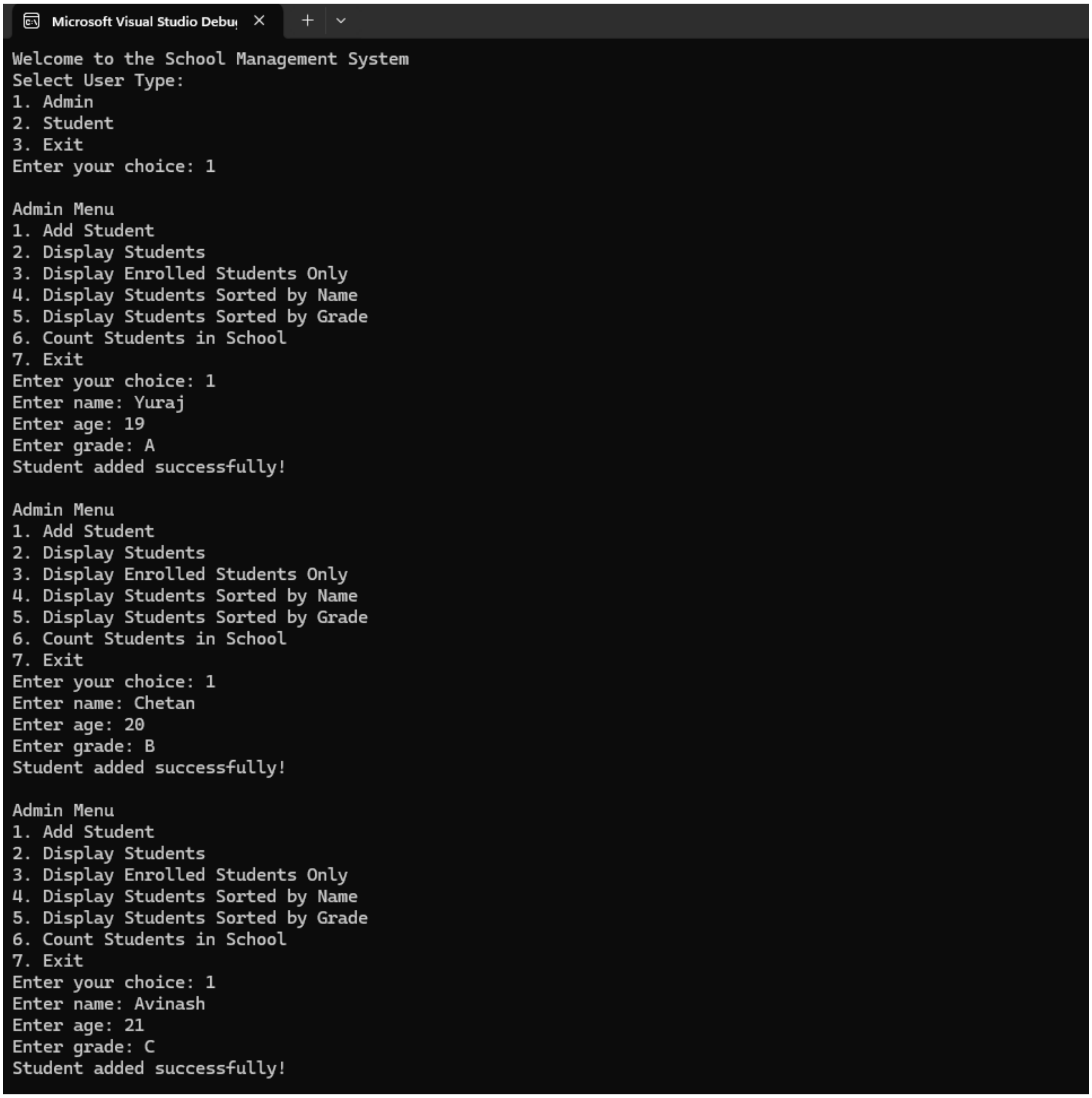
```
                case 4:
                    school.DisplayStudents(sorted: true, sortBy: "name");
                    break;
                case 5:
                    school.DisplayStudents(sorted: true, sortBy: "grade");
                    break;
                case 6:
                    Console.WriteLine($"Total students in school:
{school.GetStudentCount()}");
                    break;
                case 7:
                    Console.WriteLine("Exiting admin menu...");
                    break;
                default:
                    Console.WriteLine("Invalid choice! Please try again.");
                    break;
            }
        }
        else
        {
            Console.WriteLine("Please enter a valid number.");
        }
    } while (choice != 7);
  }
```

```
private static void AddStudentFlow(School school)
{
    Console.Write("Enter name: ");
    string name = Console.ReadLine();

    Console.Write("Enter age: ");
    if (int.TryParse(Console.ReadLine(), out int age))
    {
        Console.Write("Enter grade: ");
        string grade = Console.ReadLine();
        school.AddStudent(new Student(name, age, grade));
    }
    else
    {
        Console.WriteLine("Invalid age. Student not added.");
    }
}

private static void StudentMenu(School school)
{
    int choice;
    do
```

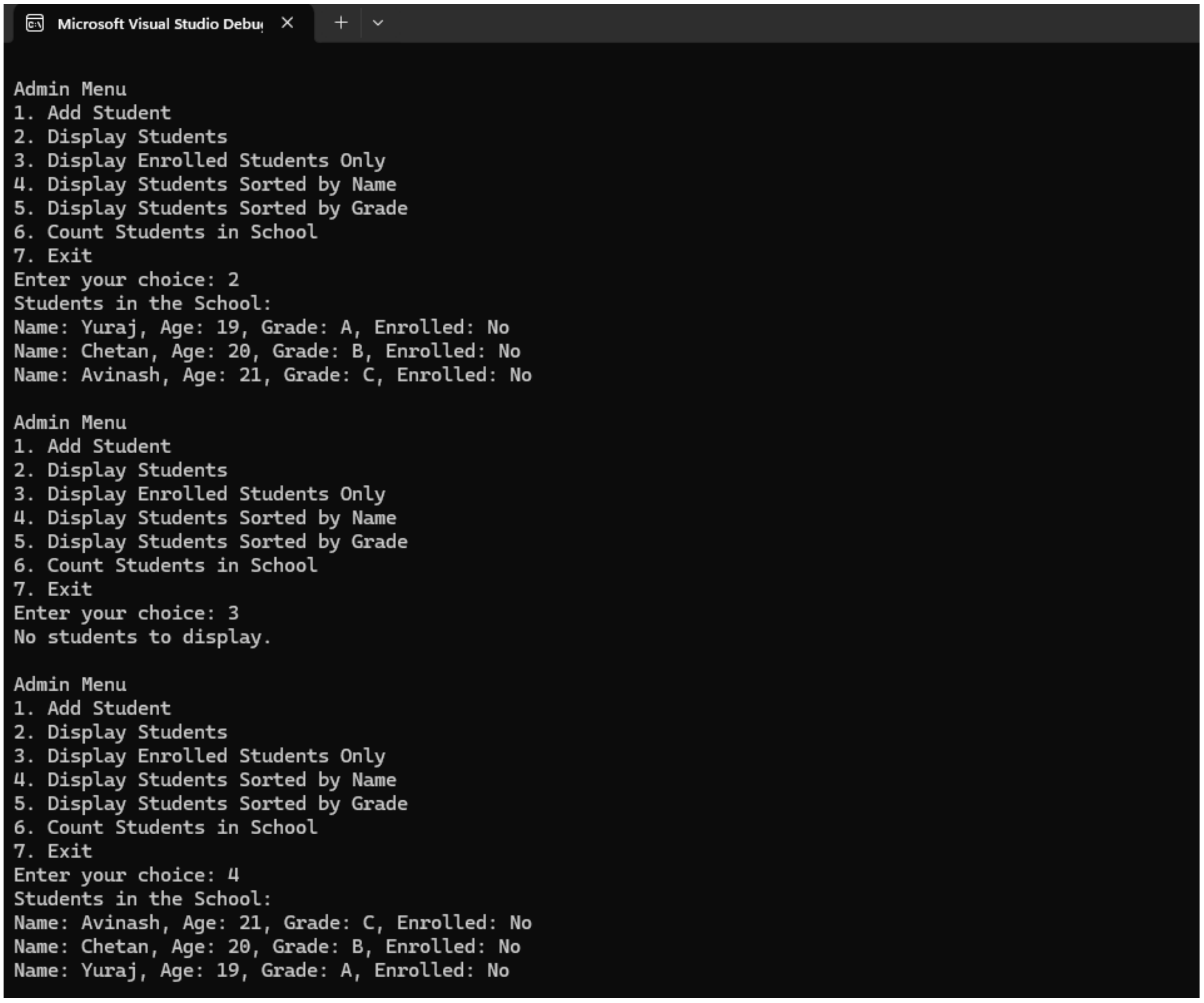
```
{
    Console.WriteLine("\nStudent Menu");
    Console.WriteLine("1. Search Student");
    Console.WriteLine("2. Enroll Student");
    Console.WriteLine("3. Unenroll Student");
    Console.WriteLine("4. View All Students (Sorted by Name)");
    Console.WriteLine("5. View All Students (Sorted by Grade)");
    Console.WriteLine("6. View Enrolled Students Only");
    Console.WriteLine("7. View Total Number of Students");
    Console.WriteLine("8. Exit");
    Console.Write("Enter your choice: ");

    if (int.TryParse(Console.ReadLine(), out choice))
    {
        switch (choice)
        {
            case 1:
                SearchStudentFlow(school);
                break;
            case 2:
                EnrollStudentFlow(school);
                break;
            case 3:
                UnenrollStudentFlow(school);
                break;
            case 4:
                school.DisplayStudents(sorted: true, sortBy: "name");
                break;
            case 5:
                school.DisplayStudents(sorted: true, sortBy: "grade");
                break;
            case 6:
                school.DisplayStudents(onlyEnrolled: true);
                break;
            case 7:
                Console.WriteLine($"Total students in school:
{school.GetStudentCount()}");
                break;
            case 8:
                Console.WriteLine("Exiting student menu...");
                break;
            default:
                Console.WriteLine("Invalid choice! Please try again.");
                break;
        }
    }
    else
```

```
        {
            Console.WriteLine("Please enter a valid number.");
        }
    } while (choice != 8);
}

private static void SearchStudentFlow(School school)
{
    Console.Write("Enter name to search: ");
    string name = Console.ReadLine();
    school.SearchStudent(name);
}

private static void EnrollStudentFlow(School school)
{
    Console.Write("Enter name to enroll: ");
    string name = Console.ReadLine();
    school.EnrollStudent(name);
}

private static void UnenrollStudentFlow(School school)
{
    Console.Write("Enter name to unenroll: ");
    string name = Console.ReadLine();
    school.UnenrollStudent(name);
}

public static void Main()
{
    var school = new School();
    int userType;

    do
    {
        Console.WriteLine("Welcome to the School Management System");
        Console.WriteLine("Select User Type:");
        Console.WriteLine("1. Admin");
        Console.WriteLine("2. Student");
        Console.WriteLine("3. Exit");
        Console.Write("Enter your choice: ");

        if (int.TryParse(Console.ReadLine(), out userType))
        {
            switch (userType)
            {
                case 1:
                    AdminMenu(school);
                    break;
                case 2:
                    StudentMenu(school);
                    break;
                case 3:
                    Console.WriteLine("Exiting the system...");
                    break;
                default:
                    Console.WriteLine("Invalid user type selected. Please try again.");
                    break;
              }
            }
            else
            {
                Console.WriteLine("Please enter a valid number.");
            }
        } while (userType != 3);
    }
}
```

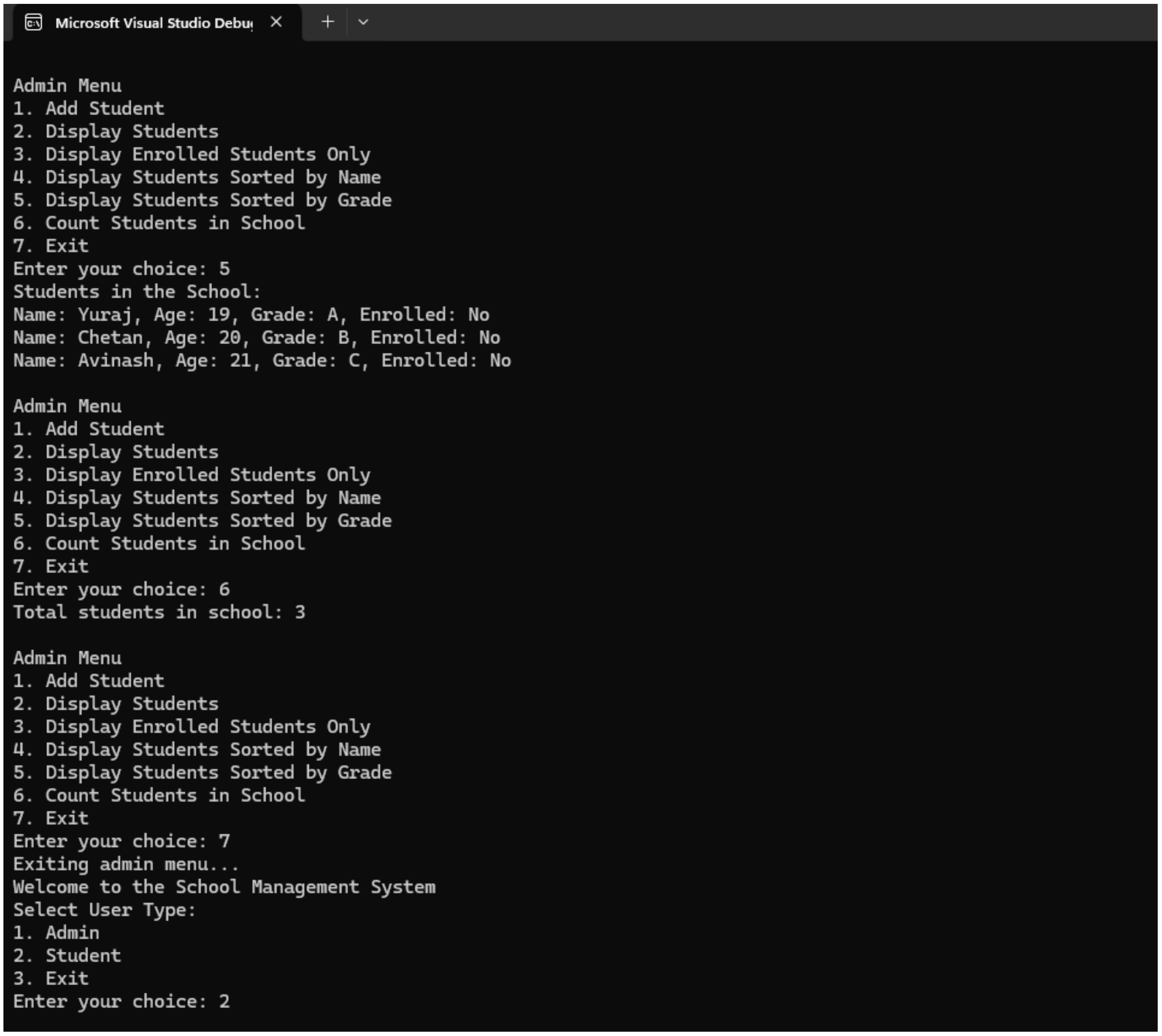
6. Result/Output/Writing Summary:



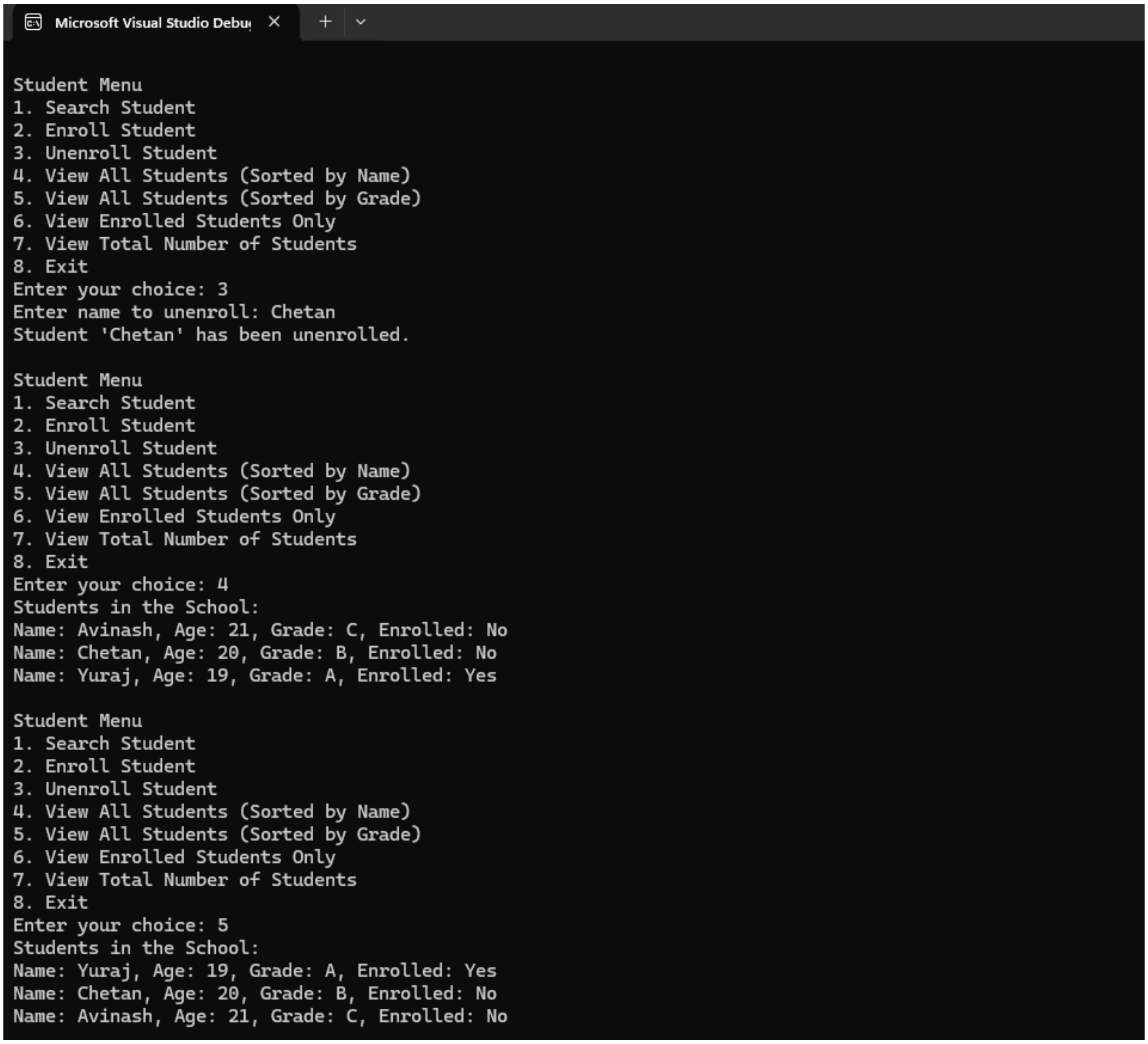
```
Welcome to the School Management System
Select User Type:
1. Admin
2. Student
3. Exit
Enter your choice: 1

Admin Menu
1. Add Student
2. Display Students
3. Display Enrolled Students Only
4. Display Students Sorted by Name
5. Display Students Sorted by Grade
6. Count Students in School
7. Exit
Enter your choice: 1
Enter name: Yuraj
Enter age: 19
Enter grade: A
Student added successfully!

Admin Menu
1. Add Student
2. Display Students
3. Display Enrolled Students Only
4. Display Students Sorted by Name
5. Display Students Sorted by Grade
6. Count Students in School
7. Exit
Enter your choice: 1
Enter name: Chetan
Enter age: 20
Enter grade: B
Student added successfully!

Admin Menu
1. Add Student
2. Display Students
3. Display Enrolled Students Only
4. Display Students Sorted by Name
5. Display Students Sorted by Grade
6. Count Students in School
7. Exit
Enter your choice: 1
Enter name: Avinash
Enter age: 21
Enter grade: C
Student added successfully!
```



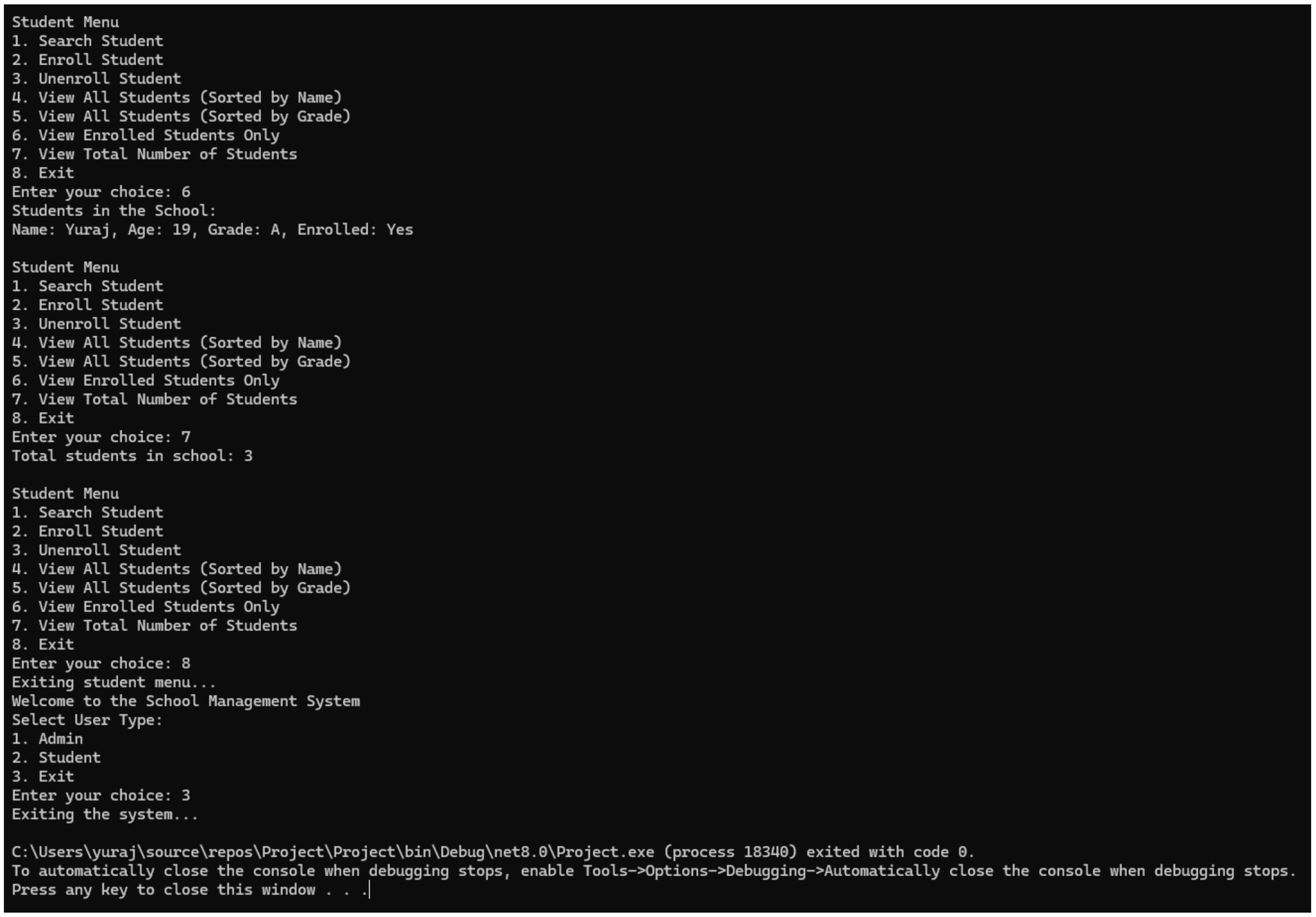
```
Admin Menu
1. Add Student
2. Display Students
3. Display Enrolled Students Only
4. Display Students Sorted by Name
5. Display Students Sorted by Grade
6. Count Students in School
7. Exit
Enter your choice: 2
Students in the School:
Name: Yuraj, Age: 19, Grade: A, Enrolled: No
Name: Chetan, Age: 20, Grade: B, Enrolled: No
Name: Avinash, Age: 21, Grade: C, Enrolled: No

Admin Menu
1. Add Student
2. Display Students
3. Display Enrolled Students Only
4. Display Students Sorted by Name
5. Display Students Sorted by Grade
6. Count Students in School
7. Exit
Enter your choice: 3
No students to display.

Admin Menu
1. Add Student
2. Display Students
3. Display Enrolled Students Only
4. Display Students Sorted by Name
5. Display Students Sorted by Grade
6. Count Students in School
7. Exit
Enter your choice: 4
Students in the School:
Name: Avinash, Age: 21, Grade: C, Enrolled: No
Name: Chetan, Age: 20, Grade: B, Enrolled: No
Name: Yuraj, Age: 19, Grade: A, Enrolled: No
```



```
Admin Menu
1. Add Student
2. Display Students
3. Display Enrolled Students Only
4. Display Students Sorted by Name
5. Display Students Sorted by Grade
6. Count Students in School
7. Exit
Enter your choice: 5
Students in the School:
Name: Yuraj, Age: 19, Grade: A, Enrolled: No
Name: Chetan, Age: 20, Grade: B, Enrolled: No
Name: Avinash, Age: 21, Grade: C, Enrolled: No

Admin Menu
1. Add Student
2. Display Students
3. Display Enrolled Students Only
4. Display Students Sorted by Name
5. Display Students Sorted by Grade
6. Count Students in School
7. Exit
Enter your choice: 6
Total students in school: 3

Admin Menu
1. Add Student
2. Display Students
3. Display Enrolled Students Only
4. Display Students Sorted by Name
5. Display Students Sorted by Grade
6. Count Students in School
7. Exit
Enter your choice: 7
Exiting admin menu...
Welcome to the School Management System
Select User Type:
1. Admin
2. Student
3. Exit
Enter your choice: 2
```



```
Student Menu
1. Search Student
2. Enroll Student
3. Unenroll Student
4. View All Students (Sorted by Name)
5. View All Students (Sorted by Grade)
6. View Enrolled Students Only
7. View Total Number of Students
8. Exit
Enter your choice: 3
Enter name to unenroll: Chetan
Student 'Chetan' has been unenrolled.

Student Menu
1. Search Student
2. Enroll Student
3. Unenroll Student
4. View All Students (Sorted by Name)
5. View All Students (Sorted by Grade)
6. View Enrolled Students Only
7. View Total Number of Students
8. Exit
Enter your choice: 4
Students in the School:
Name: Avinash, Age: 21, Grade: C, Enrolled: No
Name: Chetan, Age: 20, Grade: B, Enrolled: No
Name: Yuraj, Age: 19, Grade: A, Enrolled: Yes

Student Menu
1. Search Student
2. Enroll Student
3. Unenroll Student
4. View All Students (Sorted by Name)
5. View All Students (Sorted by Grade)
6. View Enrolled Students Only
7. View Total Number of Students
8. Exit
Enter your choice: 5
Students in the School:
Name: Yuraj, Age: 19, Grade: A, Enrolled: Yes
Name: Chetan, Age: 20, Grade: B, Enrolled: No
Name: Avinash, Age: 21, Grade: C, Enrolled: No
```



```
Student Menu
1. Search Student
2. Enroll Student
3. Unenroll Student
4. View All Students (Sorted by Name)
5. View All Students (Sorted by Grade)
6. View Enrolled Students Only
7. View Total Number of Students
8. Exit
Enter your choice: 6
Students in the School:
Name: Yuraj, Age: 19, Grade: A, Enrolled: Yes

Student Menu
1. Search Student
2. Enroll Student
3. Unenroll Student
4. View All Students (Sorted by Name)
5. View All Students (Sorted by Grade)
6. View Enrolled Students Only
7. View Total Number of Students
8. Exit
Enter your choice: 7
Total students in school: 3

Student Menu
1. Search Student
2. Enroll Student
3. Unenroll Student
4. View All Students (Sorted by Name)
5. View All Students (Sorted by Grade)
6. View Enrolled Students Only
7. View Total Number of Students
8. Exit
Enter your choice: 8
Exiting student menu...
Welcome to the School Management System
Select User Type:
1. Admin
2. Student
3. Exit
Enter your choice: 3
Exiting the system...

C:\Users\yuraj\source\repos\Project\Project\bin\Debug\net8.0\Project.exe (process 18340) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Code Explanation

Class Definitions

1. Student Class

The Student class represents individual students and includes the following properties and methods:

- **Properties**:
  - Name: Stores the name of the student.
  - Age: Stores the age of the student.
  - Grade: Stores the grade of the student.
  - IsEnrolled: A boolean indicating whether the student is currently enrolled (default is false).
- **Constructor**:
  - Takes optional parameters for name, age, and grade. These values initialize the corresponding properties when a Student object is created.
- **ToString() Method**:
  - Overrides the ToString method to provide a formatted string containing the student's details, including enrollment status.

2. School Class

The School class manages students and offers functionalities such as adding, displaying, searching, enrolling, and unenrolling students. It includes the following elements:

- **Constants**:
  - MaxStudents: The maximum number of students allowed in the school, set to 100.
- **Fields**:
  - students: A list of Student objects that stores the students currently in the school.
- **Methods**:
  - AddStudent(Student student):
    - Adds a new student to the list if the students list has fewer than MaxStudents entries.
    - Displays a message indicating whether the student was successfully added or if the school is at maximum capacity.
  - DisplayStudents(bool onlyEnrolled = false, bool sorted = false, string sortBy = "name"):
    - Displays a list of students with options to filter by enrollment status (onlyEnrolled), sort by name or grade (sorted), and specify the sorting attribute (sortBy).
    - If onlyEnrolled is true, it only displays enrolled students.
    - Sorting can be specified as either "name" or "grade".
  - SearchStudent(string name):
    - Searches for a student by name and displays their details if found, or a "not found" message otherwise.
  - EnrollStudent(string name) and UnenrollStudent(string name):
    - Toggles the enrollment status of a student based on their name.
    - If a student is found and not already in the specified enrollment state, the method changes their IsEnrolled status and provides a confirmation message.

- GetStudentCount():
    - Returns the total number of students in the school.

Menu Systems

The program has two main menu systems: **Admin Menu** and **Student Menu**. These menus allow different types of users to interact with the system's functionalities.

1. Admin Menu

The AdminMenu method displays various administrative options for managing students. Key functionalities include:

- **Add Student**:
    - Allows the admin to input details (name, age, grade) for a new student and adds them to the system using AddStudent.
- **Display Students**:
    - Shows all students in the system by calling DisplayStudents().
- **Display Enrolled Students Only**:
    - Shows only the students with IsEnrolled set to true.
- **Display Students Sorted by Name/Grade**:
    - Sorts and displays the list of students by either name or grade.
- **Count Students in School**:
    - Shows the total number of students using GetStudentCount().
- **Exit**:
    - Ends the admin session and returns to the main menu.

2. Student Menu

The StudentMenu method presents options relevant to students. Key functionalities include:

- **Search Student**:

- Searches for a student by name using SearchStudent.

- **Enroll/Unenroll Student**:
    - Allows students to change their enrollment status through EnrollStudent and UnenrollStudent.
- **View Students Sorted by Name/Grade**:
    - Displays students sorted by name or grade.
- **View Enrolled Students Only**:
    - Displays only students who are currently enrolled.
- **View Total Number of Students**:
    - Displays the current number of students using GetStudentCount().
- **Exit**:
    - Ends the student session and returns to the main menu.

Significance of Constants

- MaxStudents: Limits the maximum number of students that can be added to the school. This constant ensures that the school operates within a fixed capacity, preventing an overflow of data that could lead to memory issues or affect performance.

Error Handling

The code includes basic error handling and validations:

- **Full School**:
    - In AddStudent, if the students list has reached MaxStudents, the method displays a message indicating that the school is at full capacity.
- **Student Not Found**:
    - The SearchStudent, EnrollStudent, and UnenrollStudent methods handle cases where a student is not found in the students list, providing a clear message to the user.

- **Invalid Input**:
    - When reading user input, the int.TryParse method ensures that non-numeric inputs do not cause runtime errors. If parsing fails, the program prompts the user to enter a valid number.

This system is structured to be user-friendly, with clear options and straightforward error messages, making it a robust solution for managing student data in a school.

Evaluation Grid:

Sr. No.	Parameters	Marks Obtained	Maximum Marks
1.	Demonstration and Performance (Pre Lab Quiz)		5
2.	Worksheet		10
3.	Post Lab Quiz		5