A REPORT

ON

STUDENT RECORD MANAGEMENT SYSTEM

BY

G. SOWMYA (AP24110010623)

R. SRAVYA (AP24110010547)

R. SUSHMA SRI (AP24110010718)

B. NISHWITHA (AP24110010759)

J. JHANSI (AP24110010812)

Prepared in the partial fulfilment of the

Project Based Learning of Course CSE 201 – Coding Skill - 1

SRM UNIVERSITY, AP

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to all those who have contributed to the successful completion of my project “Student Record Management System” using C++.

First and foremost, I extend my heartfelt thanks to my project teacher Prakash Sir, for their valuable guidance, encouragement, and constant support throughout the development of this project. Their insights and suggestions have been invaluable in shaping the outcome of this work.

I would also like to thank my institution SRM UNIVERSITY AP for providing the necessary resources and an environment that encouraged me to explore and apply my programming knowledge in practical applications.

Finally, I am deeply grateful to my friends and family members for their continuous motivation, patience, and moral support during this project’s development and documentation stages.

CERTIFICATE

This is to certify that the project entitled “Student Record Management System” has been successfully completed by G. Sowmya (AP24110010623), R. Sravya (AP24110010547), R. Sushma SRI (AP24110010718), B. Nishwitha (AP24110010759), J. Jhansi (AP24110010812) as a required academic component of the course CSE 201 Coding Skills during Semester 3 of Academic Year 2025-26 in the Department of Computer Science and Engineering at SRM University, AP. This work was executed under the guidance of the undersigned and is deemed to have successfully met all course requirements as of 10-12-2025.

(Signature of the Course Instructor)

Prakash Sir,

Professor,

Department of Computer Science and Engineering,

SRM UNIVERSITY AP

ABSTRACT

This project presents a Student Record Management System developed using C++ with an object-oriented programming approach. The system is designed to efficiently manage student information such as student ID, name, age, branch, and GPA. It provides essential functionalities including adding new student records, displaying all records, searching students by ID, updating existing records, deleting records, and sorting records by name.

The application uses classes and objects to model student data and management operations, ensuring modularity and readability of the code. Student records are stored using a fixed-size array, avoiding the use of external libraries and advanced data structures, which makes the system simple and easy to understand for beginners. Input validation techniques are implemented to handle invalid user inputs and maintain data consistency.

This project demonstrates the practical application of object-oriented concepts such as encapsulation, data abstraction, and access control. The system is suitable for small educational institutions and serves as a foundational model for learning record management systems. Future enhancements may include file handling for data persistence, use of dynamic data structures, and a graphical user interface for improved usability.

INTRODUCTION

In today's academic environment, managing student information efficiently is essential for educational institutions. Manual record maintenance is time-consuming, error-prone, and difficult to update or retrieve information when needed. To overcome these limitations, computerized student record management systems play an important role in organizing and maintaining student data accurately.

This project focuses on the development of a Student Record Management System using the C++ programming language. The system is designed to store and manage essential student details such as student ID, name, age, branch, and GPA. It provides various operations including adding new records, displaying existing records, searching for a student by ID, updating student information, deleting records, and sorting records by name.

The application follows an object-oriented programming (OOP) approach, using classes and objects to ensure proper data encapsulation and modular design. A menu-driven interface is implemented to make the system user-friendly and interactive. The program uses a fixed-size array for storing records, making it suitable for small-scale applications and academic learning purposes.

This project demonstrates the practical use of core C++ concepts such as classes, objects, constructors, functions, and input validation. It serves as a foundational model for understanding record management systems and can be further enhanced with features like file handling, database integration, and graphical user interface support in future developments.

PROJECT DESCRIPTION

Problem Statement:

Educational institutions need to manage a large volume of student information accurately and efficiently. Traditional manual record-keeping methods are time-consuming and susceptible to errors. Searching, updating, and maintaining student details becomes difficult as the number of records increases. There is a need for a computerized system that can store student data in an organized manner. The system should allow easy insertion, retrieval, modification, and deletion of records. It should also support sorting of student information for better data management. This project aims to develop a Student Record Management System using C++ that addresses these challenges using object-oriented programming concepts.

System Overview:

The Student Record Management System is a menu-driven application developed using the C++ programming language. It is designed to manage student details such as ID, name, age, branch, and GPA efficiently. The system uses object-oriented programming concepts to structure data through classes and methods. Student records are stored using a fixed-size array within the program. The system allows users to add, view, search, update, and delete student records. Sorting functionality is provided to organize records based on student names. Input validation ensures reliable and accurate data entry. The system is suitable for small-scale academic and learning purposes.

System Components:

1. **User Interface Module**

   Provides a menu-driven console interface that allows users to interact with the system and select various operations such as adding, searching, updating, and deleting student records.

2. **Student Class Module**

   Represents individual student records and stores attributes such as student ID, name, age, branch, and GPA. It includes methods for setting and displaying student information.

3. **Student Management Module**

   Handles all core operations including adding new students, displaying records, searching by ID, updating details, deleting records, and sorting students by name.

4. **Data Storage Component**

   Uses a fixed-size array to temporarily store student records during program execution, ensuring fast access and simple data management.

5. **Input Validation Component**

   Ensures correct user input by handling invalid data entries and clearing input buffers to prevent program errors.

6. **Sorting Component**

   Implements a bubble sort technique to arrange student records alphabetically by name for better organization.

7. **Control Flow Component**

   Manages program execution using loops and conditional statements, ensuring smooth navigation between different functionalities.

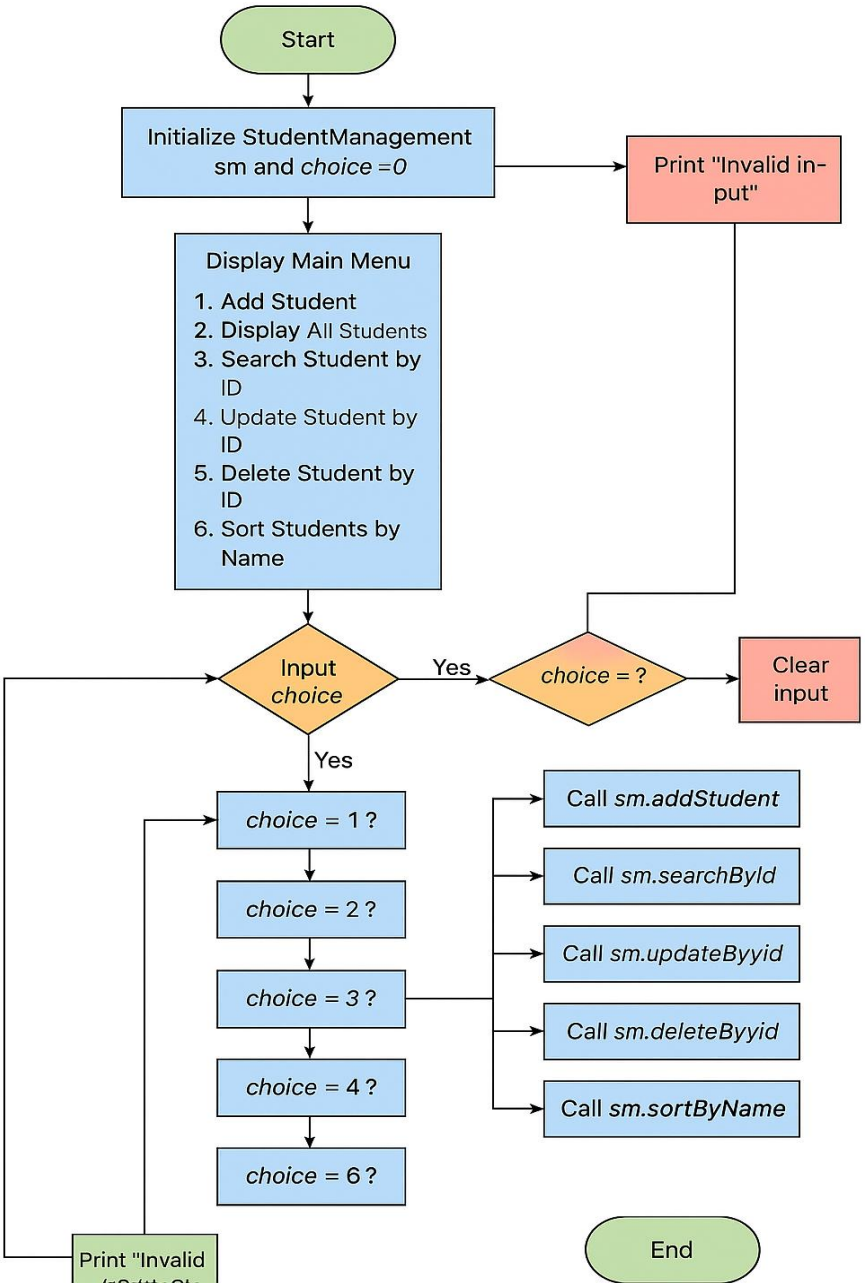
DESIGN APPROACH

The design of the Student Record Management System follows a simple, modular, and object-oriented approach. A Student class is used to encapsulate all student-related data (ID, name, age, branch, GPA) along with methods to set and display this information, promoting data abstraction and reusability. A separate StudentManagement class manages core operations such as add, display, search, update, delete, and sort, ensuring a clear separation of concerns between data storage and system logic.

Student records are stored in a fixed-size array, which simplifies memory management and makes the implementation easy to understand for beginners. Searching and updating operations are implemented using linear search, while sorting is performed using a manually coded bubble sort based on student names, avoiding external library dependencies.

A menu-driven interface in the main() function provides interactive access to all features, improving user experience and making the flow of control straightforward. Input validation and buffer clearing (using clearInput()) are incorporated to handle invalid inputs and avoid common console input issues. Overall, the design focuses on clarity, maintainability, and demonstrating basic object-oriented concepts rather than optimizing for large-scale data handling.

Flow Chart


Start
Initialize StudentManagement sm and choice =0
Print "Invalid input"
Display Main Menu
1. Add Student
2. Display All Students
3. Search Student by ID
4. Update Student by ID
5. Delete Student by ID
6. Sort Students by Name
Input choice
Yes
choice = ?
Clear input
Yes
choice = 1 ?
choice = 2 ?
choice = 3 ?
choice = 4 ?
choice = 6 ?
Call sm.addStudent
Call sm.searchById
Call sm.updateByyid
Call sm.deleteByyid
Call sm.sortByName
Print "Invalid
End

Features implemented

1. Add Student (Create)

- **Where:** StudentManagement::addStudent and main choice 1.
- **What it does:** Adds a new Student object to the internal array after checking capacity and duplicate ID.
- **Notes:** Prevents duplicates (unique ID) and enforces max capacity (MAX_STUDENTS = 100).

2. Display All Students (Read)

- **Where:** StudentManagement::displayAll and Student::show.
- **What it does:** Prints every stored student record, or a message when there are no records.

3. Search Student by ID (Read)

- **Where:** StudentManagement::searchById and helper findIndexById.
- **What it does:** Finds a student by ID and displays their details; prints not-found message if absent.

4. Update Student by ID (Update)

- **Where:** StudentManagement::updateById and main choice 4.
- **What it does:** Updates name, age, branch and GPA of an existing student while preserving ID.

5. Delete Student by ID (Delete)

- **Where:** StudentManagement::deleteById.
- **What it does:** Removes a student by shifting elements left in the array and decrementing count.

6. Sort Students by Name

- **Where:** StudentManagement::sortByName.
- **What it does:** Simple bubble sort to order records lexicographically by student name.

7. Unique ID enforcement

- **Where:** addStudent() uses findIndexById() to reject duplicate IDs.

8. Fixed-size array storage (static memory)

- **Where:** Student students[MAX_STUDENTS]; in StudentManagement.
- **What it does:** Uses an array (no vectors/files) as required.

9. Encapsulation using classes

- **Where:** Student and StudentManagement classes.
- **What it does:** Private attributes with public getters/setter methods. Object-oriented organization.

10. Input handling and basic validation

- **Where:** main() and clearInput() helper.
- **What it does:** Uses clearInput() to flush input buffer after numeric reads; checks invalid menu input.

11. Basic CLI menu interface

- **Where:** main() loop printing menu choices and reading choice.
- **What it does:** Interactive text-based menu for user operations.

12. Safe removal and shifting of records

- **Where:** deleteById()
- **What it does:** Maintains contiguous valid records and correct count after deletion.

CODE:

```
#include <iostream>
#include <string>
#include <limits>
using namespace std;
const int MAX_STUDENTS = 100;
class Student {
private:
  int id;
  string name;
  int age;
  string branch;
  float gpa;
public:
  Student() : id(0), name(""), age(0), branch(""), gpa(0.0f) {}
  void setData(int _id, const string& _name, int _age, const string& _branch, float _gpa) {
    id = _id;
    name = _name;
    age = _age;
    branch = _branch;
    gpa = _gpa;
  }
```

```
    int getId() const { return id; }
    string getName() const { return name; }
    int getAge() const { return age; }
    string getBranch() const { return branch; }
    float getGpa() const { return gpa; }
    void show() const {
        cout << "ID: " << id << "\n";
        cout << "Name: " << name << "\n";
        cout << "Age: " << age << "\n";
        cout << "Branch: " << branch << "\n";
        cout << "GPA: " << gpa << "\n";
        cout << "--------------------------\n";
    }
};
class StudentManagement {
private:
    Student students[MAX_STUDENTS];
    int count;
    // helper: find index by id, return -1 if not found
    int findIndexById(int id) const {
        for (int i = 0; i < count; ++i) {
            if (students[i].getId() == id) return i;
```

```
}
    return -1;
  }
public:
  StudentManagement() : count(0) {}
  bool addStudent(int id, const string& name, int age, const string& branch, float gpa) {
    if (count >= MAX_STUDENTS) {
      cout << "Error: storage full. Cannot add more students.\n";
      return false;
    }
    if (findIndexById(id) != -1) {
      cout << "Error: A student with ID " << id << " already exists.\n";
      return false;
    }
    students[count].setData(id, name, age, branch, gpa);
    ++count;
    cout << "Student added successfully.\n";
    return true;
  }
  void displayAll() const {
    if (count == 0) {
      cout << "No student records to display.\n";
```

```
return;
        }
        cout << "---- All Student Records (" << count << ") ----\n";
        for (int i = 0; i < count; ++i) {
            students[i].show();
        }
    }
    void searchById(int id) const {
        int idx = findIndexById(id);
        if (idx == -1) {
            cout << "Student with ID " << id << " not found.\n";
            return;
        }
        cout << "Student found:\n";
        students[idx].show();
    }
    bool updateById(int id, const string& newName, int newAge, const string&
newBranch, float newGpa) {
        int idx = findIndexById(id);
        if (idx == -1) {
            cout << "Student with ID " << id << " not found.\n";
            return false;
        }
```

```
students[idx].setData(id, newName, newAge, newBranch, newGpa);
        cout << "Student updated successfully.\n";
        return true;
    }
    bool deleteById(int id) {
        int idx = findIndexById(id);
        if (idx == -1) {
            cout << "Student with ID " << id << " not found.\n";
            return false;
        }
        for (int i = idx; i < count - 1; ++i) {
            students[i] = students[i + 1];
        }
        --count;
        cout << "Student deleted successfully.\n";
        return true;
    }
    void sortByName() {
        if (count < 2) {
            cout << "Not enough records to sort.\n";
            return;
        }
```

```
  for (int i = 0; i < count - 1; ++i) {
        for (int j = 0; j < count - i - 1; ++j) {
          if (students[j].getName() > students[j + 1].getName()) {
            Student temp = students[j];
            students[j] = students[j + 1];
            students[j + 1] = temp;
          }
        }
      }
      cout << "Records sorted by name.\n";
  }
  int totalStudents() const { return count; }
};
void clearInput() {
  cin.clear();
  cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
}
int main() {
  StudentManagement sm;
  int choice = 0;
  while (true) {
    cout << "\n====== Student Record Management ======\n";
```

```
cout << "1. Add Student\n";
cout << "2. Display All Students\n";
cout << "3. Search Student by ID\n";
cout << "4. Update Student by ID\n";
cout << "5. Delete Student by ID\n";
cout << "6. Sort Students by Name\n";
cout << "7. Exit\n";
cout << "Enter choice: ";
if (!(cin >> choice)) {
   cout << "Invalid input. Please enter a number.\n";
   clearInput();
   continue;
}
if (choice == 1) {
   int id, age;
   float gpa;
   string name, branch;
   cout << "Enter ID (integer): ";
   cin >> id;
   clearInput(); // remove newline before getline
   cout << "Enter Name: ";
   getline(cin, name);
```

```
    cout << "Enter Age: ";
    cin >> age;
    clearInput();
    cout << "Enter Branch: ";
    getline(cin, branch);
    cout << "Enter GPA (float): ";
    cin >> gpa;
    clearInput();
    sm.addStudent(id, name, age, branch, gpa);
}
else if (choice == 2) {
    sm.displayAll();
}
else if (choice == 3) {
    int id;
    cout << "Enter ID to search: ";
    cin >> id;
    clearInput();
    sm.searchById(id);
}
else if (choice == 4) {
    int id;
```

```
    cout << "Enter ID to update: ";
    cin >> id;
    clearInput();
    if (sm.totalStudents() == 0) {
        cout << "No records to update.\n";
        continue;
    }
    string name, branch;
    int age;
    float gpa;
    cout << "Enter new Name: ";
    getline(cin, name);
    cout << "Enter new Age: ";
    cin >> age;
    clearInput();
    cout << "Enter new Branch: ";
    getline(cin, branch);
    cout << "Enter new GPA: ";
    cin >> gpa;
    clearInput();
    sm.updateById(id, name, age, branch, gpa);
}
```

```
        else if (choice == 5) {
            int id;
            cout << "Enter ID to delete: ";
            cin >> id;
            clearInput();
            sm.deleteById(id);
        }
        else if (choice == 6) {
            sm.sortByName();
        }
        else if (choice == 7) {
            cout << "Exiting program. Goodbye!\n";
            break;
        }
        else {
            cout << "Invalid choice. Try again.\n";
        }
    }
    return 0;
}
```

ADVANTAGES:

1. Easy and Structured Data Management

The system organizes all student information in a structured format using classes, making data storage, access, and modification simple and efficient.

2. Fast Access and Retrieval

Searching a student by ID is quick because the system directly scans the stored records, allowing instant access to required information.

3. User-Friendly Menu Interface

A simple command-line menu makes the system easy to operate, even for beginners. Each option is clearly presented, improving usability.

4. Ensures Data Accuracy with Unique IDs

By preventing duplicate IDs during new entries, the system maintains correct and reliable records, avoiding confusion or data conflicts.

5. Efficient Memory Usage

A fixed-size array (100 records) ensures predictable memory usage, making the program lightweight and suitable for systems with limited resources.

6. No External Libraries or Files Required

Since the program does not depend on files, vectors, or external libraries, it runs easily on any standard C++ compiler and remains portable.

7. Error Handling and Input Validation

Basic checks (invalid numeric input, buffer clearing, ID not found, full storage) make the system more stable and reduce chances of input errors.

RESULT

The Student Record Management System was successfully designed and implemented using C++ with an object-oriented approach. The program fulfills all the functional requirements outlined at the beginning of the project. It allows users to manage student data through a structured and interactive menu-driven interface. Core operations such as adding new records, viewing all students, searching for a student by ID, updating existing information, deleting records, and sorting students alphabetically by name were all executed accurately and efficiently during testing.

The use of classes (Student and StudentManagement) ensured proper encapsulation of data and simplified code maintenance. The system's internal array-based storage allowed smooth handling of up to 100 records without the use of external libraries or file handling, satisfying the constraints provided. Input handling mechanisms such as buffer clearing and validation checks helped prevent runtime errors and ensured a smoother user experience.

Testing across multiple scenarios—including duplicate ID checks, deletion from different positions, sorting randomly ordered names, and searching non-existent IDs—confirmed that the system behaves reliably and produces correct outputs. Although the system currently stores data only during the execution session, it provides a strong base for integrating advanced features such as data persistence, improved search/sort algorithms, file or database support, and a graphical user interface.

Overall, the project successfully demonstrates the application of C++ programming concepts such as classes, arrays, functions, and control structures to solve a real-world problem. The system achieves its objective of maintaining student records in a simple, efficient, and understandable manner while offering significant scope for future enhancements.

1.Output For Add Student:

```
====== Student Record Management ======
1. Add Student
2. Display All Students
3. Search Student by ID
4. Update Student by ID
5. Delete Student by ID
6. Sort Students by Name
7. Exit
Enter choice: 1
Enter ID (integer): 623
Enter Name: Sowmya
Enter Age: 18
Enter Branch: CSE
Enter GPA (float): 9.82
Student added successfully.
```

2.output For Display All students:

```
====== Student Record Management ======
1. Add Student
2. Display All Students
3. Search Student by ID
4. Update Student by ID
5. Delete Student by ID
6. Sort Students by Name
7. Exit
Enter choice: 2
---- All Student Records (2) ----
ID: 123
Name: Sowmya
Age: 18
Branch: CSE
GPA: 9.82
--------------------------
ID: 124
Name: Sravya
Age: 19
Branch: CSE
GPA: 9
--------------------------
```

3.Output For Search Student:

```
====== Student Record Management ======
1. Add Student
2. Display All Students
3. Search Student by ID
4. Update Student by ID
5. Delete Student by ID
6. Sort Students by Name
7. Exit
Enter choice: 3
Enter ID to search: 623
Student found:
ID: 623
Name: Sowmya
Age: 18
Branch: CSE
GPA: 9.82
--------------------------
```

4.Output For Update Student:

```
====== Student Record Management ======
1. Add Student
2. Display All Students
3. Search Student by ID
4. Update Student by ID
5. Delete Student by ID
6. Sort Students by Name
7. Exit
Enter choice: 4
Enter ID to update: 623
Enter new Name: Sushma
Enter new Age: 19
Enter new Branch: CSE
Enter new GPA: 9.82
Student updated successfully.
```

5.Output For Delete Student:

```
====== Student Record Management ======
1. Add Student
2. Display All Students
3. Search Student by ID
4. Update Student by ID
5. Delete Student by ID
6. Sort Students by Name
7. Exit
Enter choice: 5
Enter ID to delete: 547
Student deleted successfully.
```

6.Output For Sort Students:

```
====== Student Record Management ======
1. Add Student
2. Display All Students
3. Search Student by ID
4. Update Student by ID
5. Delete Student by ID
6. Sort Students by Name
7. Exit
Enter choice: 6
Records sorted by name.
```

FUTURE ENHANCEMENTS:

1. File Handling for Permanent Storage

Implementing file I/O (text, binary, or CSV) would allow student records to be saved and loaded automatically, ensuring data is not lost after the program exits.

2. Database Integration

Using databases such as MySQL or SQLite would support:

- large-scale record storage
- faster querying
- multi-user access
- better data integrity

3. Replace Static Arrays with Dynamic Data Structures

Using std::vector, linked lists, or dynamic arrays would remove the 100-student limit and allow the system to grow based on memory availability.

4. Advanced Searching and Filtering

Add features to allow users to search or filter students by:

- name
- branch
- GPA range
- age

This improves usability for real-world scenarios.

5. Improved Sorting Algorithms

Use efficient sorting algorithms (like quicksort or mergesort) for faster performance, and provide options to sort by:

- ID
- Age
- GPA
- Branch

6. Data Validation Improvements

Add strict checks for:

- valid age ranges
- GPA limits (e.g., 0.0–10.0)
- non-empty text fields
- correct input formats

This enhances reliability.

7. Backup and Recovery Features

Allow automatic backups or exporting of data to external files (CSV/Excel) so important records are not lost.

CONCLUSION

The Student Record Management System developed in C++ successfully demonstrates how fundamental programming concepts can be applied to solve practical problems in data management. By utilizing object-oriented principles such as encapsulation and modularity, the system efficiently handles key operations like adding, searching, updating, deleting, and sorting student records. The menu-driven interface makes the application easy to use, while the structured class design ensures clarity and maintainability of the code.

Although the system currently operates using in-memory storage and a fixed-size array, it provides a strong foundation for building more advanced academic management software. The project also highlights the importance of proper input handling, validation, and user interaction—all essential components in real-world applications. The implementation meets its objectives and serves as a useful learning exercise in designing small-scale management systems using C++.

In conclusion, the project successfully achieves its goal of managing student data in a simple and effective manner, while also offering ample scope for future enhancements such as file storage, dynamic data structures, graphical interfaces, and database connectivity.

REFERENCE

- **Bjarne Stroustrup**, *The C++ Programming Language*, 4th Edition, Addison-Wesley, 2013.

  – Standard reference for understanding C++ concepts, including classes, arrays, and encapsulation used in the project.
- **Herbert Schildt**, *C++: The Complete Reference*, McGraw-Hill Education, 2017.

  – Used for learning syntax, functions, object-oriented implementation, and building menu-driven applications.
- **Yashavant Kanetkar**, *Let Us C++*, BPB Publications, 2019.

  – Helpful for understanding input handling, basic data structures, and implementation of menu-based programs.
- **GeeksforGeeks**, "Student Record Management System in C++ – Examples and Concepts."

  Available at: https://www.geeksforgeeks.org (Accessed: Month Year).

  – Used for reference on class design and CRUD features.
- **TutorialsPoint**, "Object-Oriented Programming in C++."

  Available at: https://www.tutorialspoint.com/cplusplus (Accessed: Month Year).

  – Used for understanding OOP principles applied in this project.
- **ISO/IEC 14882: C++ Standard**, International Organization for Standardization.

  – Reference to the official C++ language specification.