National College of Computer studies

Paknajol,Kathmandu

Report on

University Management System

using C++ language

Submitted by: Submitted to:

Anshu Hada Janak Joshi

Kamana Shrestha

Programme: Bsc CSIT 2ndsem

Section: A

Table of Content

[Introduction 2](#_Toc183963479)

[Background and Objectives 2](#_Toc183963480)

[Overview of structure of report 2](#_Toc183963481)

[Project Description 4](#_Toc183963482)

[Purpose and scope 4](#_Toc183963483)-5

[Methodology 6](#_Toc183963485)

[Development Process 6](#_Toc183963486)

[Tools and Resources 6](#_Toc183963487)

[Results and Discussion 7](#_Toc183963488)

[Presentation of Project Outcomes and Achievements 7](#_Toc183963489)-8

[Discussion of Challenges Faced and Lessons Learned 7](#_Toc183963490)

[Conclusion 9](#_Toc183963491)

[Future Works 10](#_Toc183963492)

[References 11](#_Toc183963493)

[Appendices 12](#_Toc183963494)-24

Abstract

University Management System Project is a C++ based application designed to streamline and automate various administrative tasks within a university. This application written in the C++ programming language, allows users to generate, search, update, and delete. The system has a user-friendly interface that allows users to enter and save student information like roll numbers, names, course address and grades obtained. File handling technologies are used to effectively store and retrieve student records. The system intends to improve the efficiency and accuracy of university management operations. This study summarizes the University Management System, including its architecture, functions, implementation details, and prospective areas for future improvement.

# Introduction

In today’s digital age, universities are increasingly adopting technology to streamline their operations and enhance the overall educational experience. This University Management System (UMS) project, developed using C++, aims to address the growing need for efficient and automated administration of various university functions.

## Background and Objectives

Traditional University Management System involves manual processes that are prone to errors and time-consuming. Recognizing these limitations, this project seeks to develop a user-friendly and secure management system.

This UMS project aims to,

* Automate administrative tasks
* Provide a user-friendly interface
* Centralize data management

### Overview of structure of report

a) Introduction: Provides background information on the University Management System project, outlining its objectives and setting the report’s context.

b) Background: Examines the issues of manual university management, emphasizing the need for automated alternatives.

c) Objectives: Outline the particular goals that will guide the development of the University Management System.

d) Methodology: Discusses the methodology and procedures utilized to construct the system.

e) System Overview: Provides a detailed description of the University Management System, including its architecture, functionalities, and user interface.

f) Results and Discussion: The system testing and evaluation results are presented,

as well as a discussion of the findings and insights.

g) Future Work: Explores potential areas for system refinement and development.

h) Conclusion: Summarizes the report’s important findings and contributions, emphasizes the importance of the UniversityManagement System, and provides concluding remarks.

i) References: A list of all sources cited in the report.

# Project Description

## Purpose and scope

The primary purpose of a University Management System aims to improve efficiency, accuracy and accessibility of student, faculty and course information. The scope of UMS project is for record maintenance and registration.

### Problems being solved

Traditional manual methods of maintaining student records can be time-consuming, error-prone, and inefficient. The system automates the process of adding, searching, updating, and deleting student information, reducing the chances of human error. As the number of students increases, manual record-keeping becomes increasingly challenging. The system can handle a large number of student records efficiently, making it scalable for growing institutions.

#### Functionality and features

1. Student Management**:**
   * Add Student**:** Allows users to input student details.
   * Search Student: Searches for a student by roll number.
   * Update Student: Updates the information of an existing student.
   * Delete Student: Deletes a student record.
   * Display All Students: Displays a list of all students.
2. Data Persistence**:**
   * Save Data to File: Saves student data to a text file.
   * Load Data from File: Loads student data from a text file.

# Methodology

## Development Process

* Requirements Gathering: First of all, we gathered all the information related to the problem and what is required to solve the problem.
* Design: In the next step, we visualized a general system architecture such as the user interface and other system components. After that we visualized identifying the core functionalities like add, search, update, and delete students record then display all students record and at last save and load data from a file.
* Implementation: Now, We translated the design into code. This involved writing the code using C++ programming language in Dev-C++.
* Testing: After writing the code, we tested the application to check if all the components work properly. Through testing, we could make sure that the system meets all the requirements and solves the problem in an efficient manner.
* Deployment: Finally, after making sure that there are no bugs or errors, the system is ready to be used in the university.

### Tools and Resources

* IDE (Integrated Development Environment): The project used Dev-C++ IDE to write, edit, and debug code.
* Libraries: The application used libraries such as the Standard C++ Library like iostream, fstream, string, iomanip, and windows.h for input/output, file handling, string manipulation, formatting, and platform-specific operations.
* Data Structures**:** The system utilizes a Student class to store information about each student, including roll number, name, course, address, and grade.
* File Handling: Data is persisted to a text file named "students.txt" for long-term storage.
* Modularization: The system is divided into well-defined functions for each operation, promoting code reusability and maintainability.

# Results and Discussion

## Presentation of Project Outcomes and Achievements

The creation of the University Management System (UMS) has yielded notable outcomes and accomplishments, hence contributing to the improvement in management system. Key results include:

* Automation of University Management: The UMS effectively automates important management functions such as generate, display, search, and deletion. This automation has simplified administrative operations, decreasing manual work and saving time for administrators and faculty members.
* Enhanced Efficiency and Accuracy: By automating data entry and computation methods, the UMS has enhanced processing efficiency and accuracy. The system ensures that student’s data is accurate and dependable, while also minimizing errors.
* Enhanced Accessibility: Administrators, faculty members mayeasily record student’s information thanks to the UMS’s user-friendly design.

### Discussion of Challenges Faced and Lessons Learned

Several problems emerged during the project, providing great learning experiences:

* Technical Challenges: During the implementation phase, some technical issues arose, such as troubleshooting complex code and optimizing system performance. These issues required extensive problem solving and asking help from friends and looking for solutions on the internet.
* User Feedback: Without proper feedback from other users, the system could not be trusted for solving the problem. That is why, we asked many of my friends to run their code in their computer and ask their feedback.

By encountering these problems, I was able to learn the following lessons:

* Adaptability: Projects are frequently met with unexpected challenges or changes in needs. Being flexible and quick in our response to these issues is critical for overcoming obstacles and ensuring project success.
* Time Management: Effective time management is essential for fulfilling project deadlines and milestones. Effective task planning, scheduling, and prioritization can help us manage your time more efficiently.
* Continuous Improvement: Reflecting on project successes and failures might

yield useful insights for future projects. Embracing a culture of continuous

development, learning from prior events, and putting those lessons into practice can lead to improved results in future attempts.

# Conclusion

In conclusion, the University Management System project provides a robust solution for managing student records in a university. It offers a user-friendly interface, efficient data handling, and data persistence capabilities. By incorporating advanced features and a graphical user interface in future iterations, the system can be further enhanced to meet the evolving needs of educational institutions. The system’s capacity to automate management operations, increase productivity, and improve accuracy illustrates its potential to address common difficulties confronting educational institutions. Furthermore, the UMS’s success demonstrates the revolutionary potential of technology in the field of education. By using technology to optimize administrative operations, minimize manual workload, and improve overall management system.

# Future Works

To further enhance the system, the following areas can be explored:

* Graphical User Interface (GUI): Developing a GUI can significantly improve user experience and make the system more accessible to a wider range of users.
* Database Integration: Incorporating a database management system (DBMS) can improve data organization, retrieval, and scalability.
* Advanced Features: Implementing additional features such as fee management, course enrollment, and exam scheduling can make the system more comprehensive.
* Security: Implementing security measures, such as user authentication and data encryption, can protect sensitive student information.
* Error Handling and Exception Handling**:** Enhancing error handling and exception handling mechanisms can make the system more robust and user-friendly.
* Testing and Debugging: Continuous testing and debugging can help identify and fix issues, ensuring the system's reliability.
* Documentation: Creating clear and concise documentation can aid in understanding, maintenance, and future development.

By addressing these areas, the University Management System can evolve into a more powerful and versatile tool for managing university operations.

# References

1. <https://www.javatpoint.com/university-management-system-in-cpp>
2. <https://github.com/alii13/University-Management-System>
3. <https://www.youtube.com/watch?v=zMien1r25xQ>

# Appendices

Source Code:

#include <iostream>

#include <fstream>

#include <string>

#include <iomanip> // for formatting

#include <windows.h> // For Windows sleep

using namespace std;

class Student {

public:

string name, course, address;

int rollNo;

float grade;

void input()

{

cout << "\tEnter student details:\n";

cout << "\tRoll No: ";

cin >> rollNo;

cin.ignore(); // To clear the buffer before reading the next line

cout << "\tName: ";

getline(cin, name);

cout << "\tCourse: ";

getline(cin, course);

cout << "\tAddress: ";

getline(cin, address);

cout << "\tGrade: ";

cin >> grade; // Finally, enter grade

}

void display()

{

cout << "\t---------------------------------------------\n";

cout << "\tRoll No: " << setw(5) << rollNo << endl;

cout << "\tName: " << setw(15) << name << endl;

cout << "\tCourse: " << setw(10) << course << endl;

cout << "\tAddress: " << setw(15) << address << endl;

cout << "\tGrade: " << setw(10) << fixed << setprecision(2) << grade << endl;

cout << "\t---------------------------------------------\n";

system("pause");

}

};

void addStudent(Student s[], int &n)

{

if (n < 100)

{

s[n].input();

n++;

cout << "\tStudent added successfully.\n";

cout << "\tPausing for a moment...\n";

Sleep(2000); // Sleep for 2 seconds (2000 milliseconds)

char continueAdding;

cout << "\tDo you want to add another student? (Y/N): ";

cin >> continueAdding;

cin.ignore(10000, '\n'); // To clear the input buffer after reading continueAdding

// If user enters 'Y' or 'y', continue adding students

if (continueAdding == 'Y' || continueAdding == 'y')

{

system("cls");

addStudent(s, n); // Call addStudent recursively to add another student

}

}

else

{

cout << "Maximum number of students reached.\n";

}

}

void searchStudent(Student s[], int n, int rollNo) {

bool found = false;

for (int i = 0; i < n; i++) {

if (s[i].rollNo == rollNo) {

s[i].display();

found = true;

break;

}

}

if (!found) {

cout << "\tStudent not found.\n";

}

cin.ignore(10000, '\n');

}

void updateStudent(Student s[], int n, int rollNo) {

bool found = false;

for (int i = 0; i < n; i++) {

if (s[i].rollNo == rollNo) {

s[i].input();

found = true;

break;

}

}

if (!found) {

cout << "\tStudent not found.\n";

}

}

void deleteStudent(Student s[], int &n, int rollNo) {

bool found = false;

for (int i = 0; i < n; i++) {

if (s[i].rollNo == rollNo) {

for (int j = i; j < n - 1; j++) {

s[j] = s[j + 1];

}

n--;

found = true;

cout << "\tStudent deleted.\n";

system("pause");

break;

}

}

if (!found) {

cout << "\tStudent not found.\n";

system("pause");

}

}

void displayAllStudents(Student s[], int n) {

cout << "\tList of All Students:\n";

for (int i = 0; i < n; i++) {

s[i].display();

}

}

void saveDataToFile(Student s[], int n) {

ofstream file("students.txt");

if (file.is\_open()) {

for (int i = 0; i < n; i++) {

file << s[i].rollNo << endl; // Save roll number first

file << s[i].name << endl;

file << s[i].course << endl;

file << s[i].address << endl;

file << s[i].grade << endl;

}

file.close();

cout << "\tData saved to file.\n";

} else {

cout << "\tError saving data to file.\n";

}

}

void loadDataFromFile(Student s[], int &n) {

ifstream file("students.txt");

if (file.is\_open()) {

int i = 0;

while (file >> s[i].rollNo >> ws) { // ws ignores whitespace

getline(file, s[i].name);

getline(file, s[i].course);

getline(file, s[i].address);

file >> s[i].grade;

i++;

}

n = i;

file.close();

cout << "\tData loaded from file.\n";

} else {

cout << "\tError loading data from file or file is empty.\n";

}

}

int main() {

const int MAX\_STUDENTS = 100;

Student students[MAX\_STUDENTS];

int numStudents = 0;

loadDataFromFile(students, numStudents); // Load existing data from file

int choice;

do {

system("cls");

cout << "\tWelcome To University Management System" << endl;

cout << "\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\t1. Add Student."<< endl;

cout << "\t2. Search Student."<< endl;

cout << "\t3. Update Student."<< endl;

cout << "\t4. Delete Student."<< endl;

cout << "\t5. Display All Students."<< endl;

cout << "\t6. Save Data to File."<< endl;

cout << "\t7. Exit."<< endl;

cout << "\tEnter your choice: ";

cin >> choice;

switch (choice) {

case 1:

system("cls");

addStudent(students, numStudents); // Add student and handle continuation

break;

case 2: {

system("cls");

int rollNo;

cout << "\tEnter roll number to search: ";

cin >> rollNo;

cin.ignore(10000, '\n');

searchStudent(students, numStudents, rollNo);

system("pause");

break;

}

case 3: {

system("cls");

int rollNo;

cout << "\tEnter roll number to update: ";

cin >> rollNo;

updateStudent(students, numStudents, rollNo);

system("pause");

break;

}

case 4: {

system("cls");

int rollNo;

cout << "\tEnter roll number to delete: ";

cin >> rollNo;

deleteStudent(students, numStudents, rollNo);

break;

}

case 5:

system("cls");

displayAllStudents(students, numStudents);

system("pause");

break;

case 6:

system("cls");

saveDataToFile(students, numStudents);

break;

case 7:

system("cls");

cout << "\tExiting...\n";

// Sleep for 2 seconds before exiting

Sleep(2000);

break;

default:

cout << "\tInvalid choice.\n";

}

} while (choice != 7);

return 0;

}

Output:

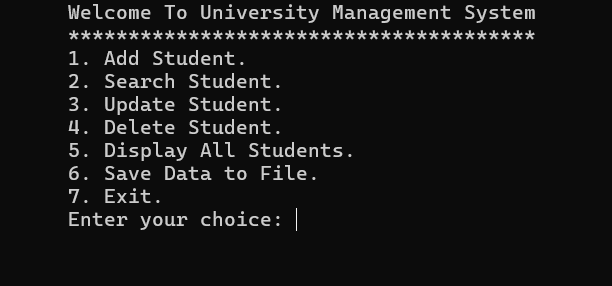


Fig 1.1: Home Interface

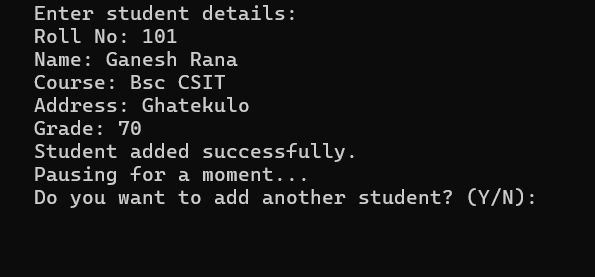


Fig 1.2: Option 1 (Input Details)

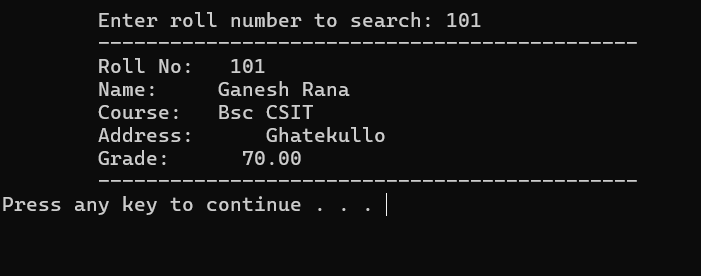


Fig 1.3: Option 2 (Search Data)

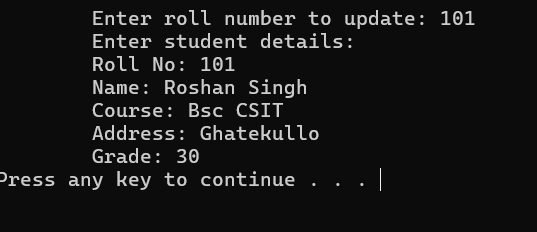


Fig 1.4: Option 3 (Update Data)

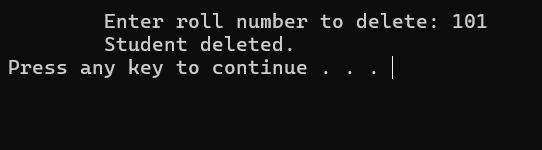
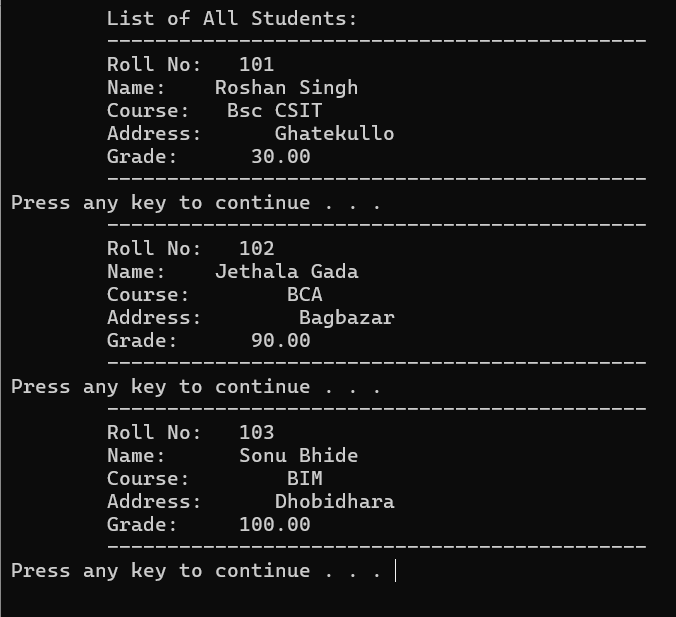


Fig 1.5: Option 4 (Delete Data)



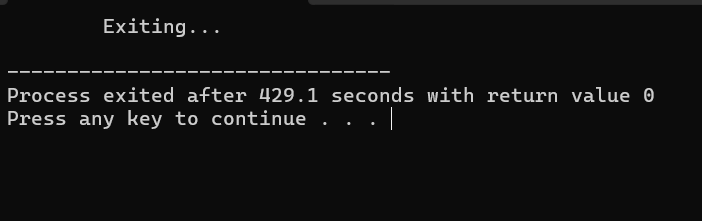
Fig 1.6: Option 5 (Display Data)

Fig 1.7: Option 7 (Exit)