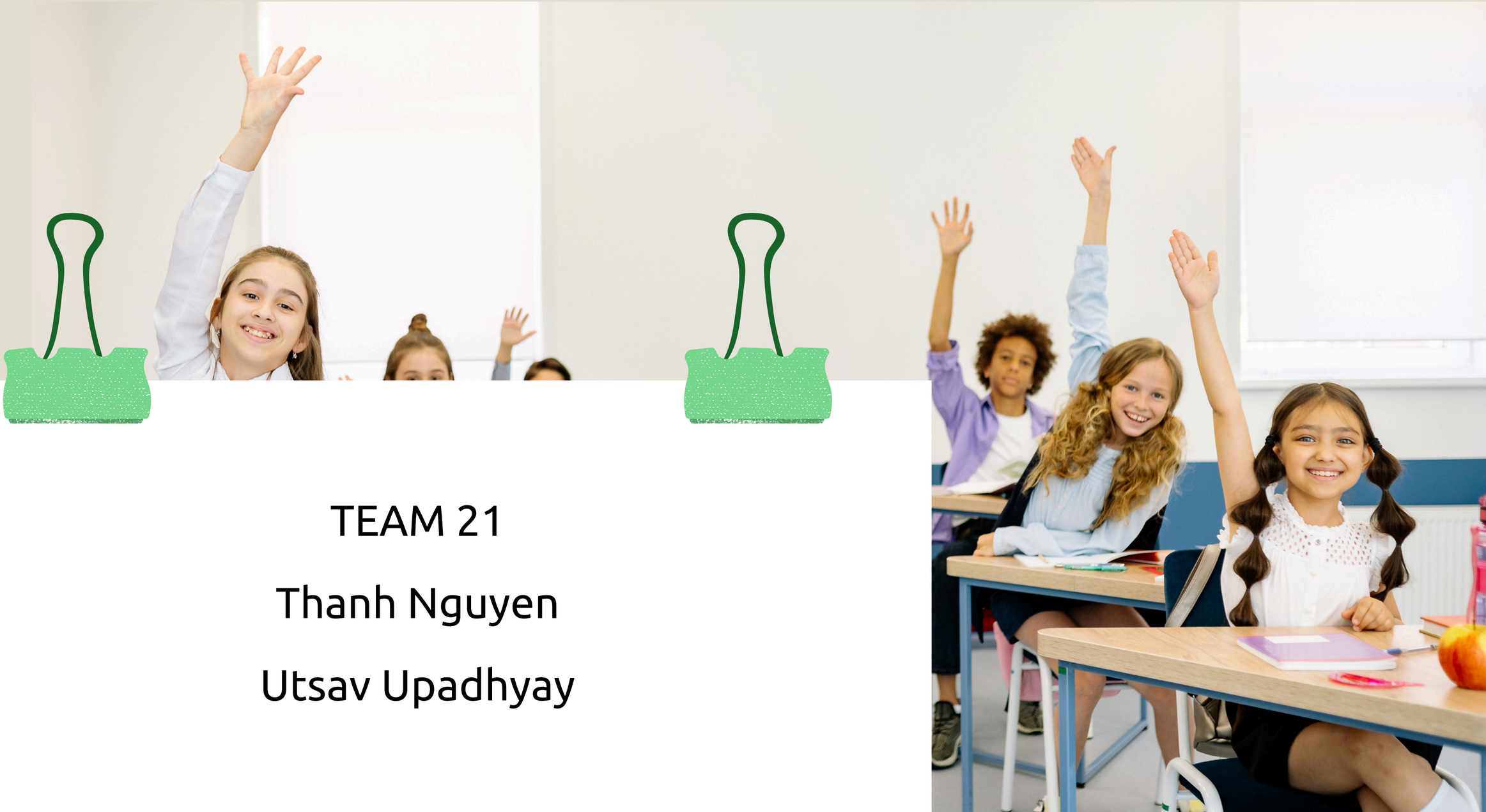
CS2263 - SYSTEM SOFTWARE DEVELOPMENT

STUDENT GRADE MANAGEMENT SYSTEM



TEAM 21

Thanh Nguyen

Utsav Upadhyay

INTRODUCTION

Problem: Manual grade tracking is error-prone and inefficient.

So we built an system to manage students, grades, and everything in between :)
A menu-driven application that supports adding, searching, displaying, sorting, and removing students, with persistent storage.

Target Users: Small academic institutions or teaching assistants

Our Features

Display Students

View Profiles

GPA Calculatic

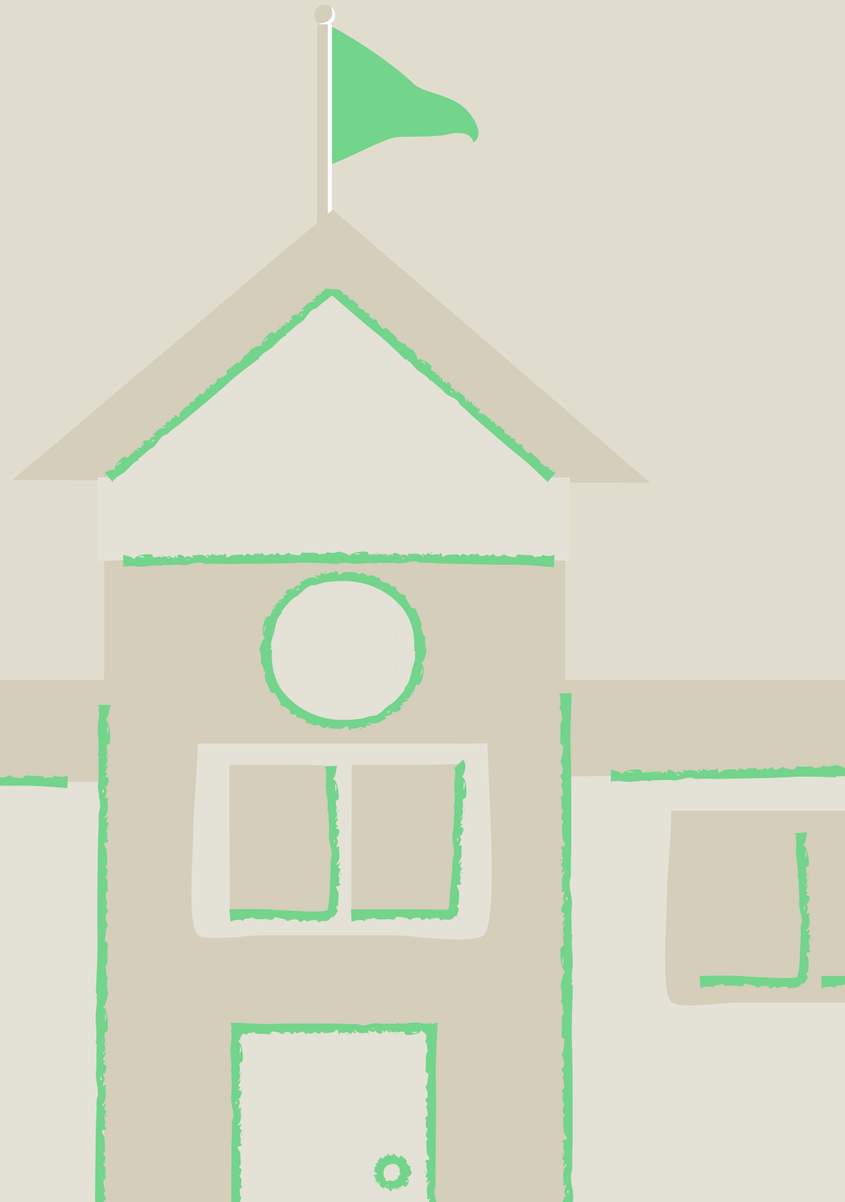
Partial Search

Performance Bar chart

New Entries

STUDENT GRADE MANAGEMENT SYSTEM

DEMONSTRATION



Program Implementation

MAIN.C

On execution of main.c, all corresponding data from csv files is loaded into the two linked lists.

(dynamically allocated memory for each node)

ACTION

EITHER the lists are simply used to retrieve and display data.

OR new data (NODE) is added or deleted

SAVE

IF node(s) are added/deleted -> the updated lists are written in the csv files to save data.

FREE

Once the user Selects EXIT, the allocated memory is recursively freed to prevent leaks

CONCEPTS USED

RECURSION

DATA STRUCTURES

FILE I/O

LINKED LISTS

DYNAMIC MEMORY ALLOCATIO

HOW AND WHERE?

Programming language: C

Our whole project is based on this language :)

FILE I/O

We used file I/O to read from and write data to two CSV files. This allows the system to retain student and grade data between sessions. At the start of the program, it loads existing data into linked lists, and saves all updated records when a node is modified

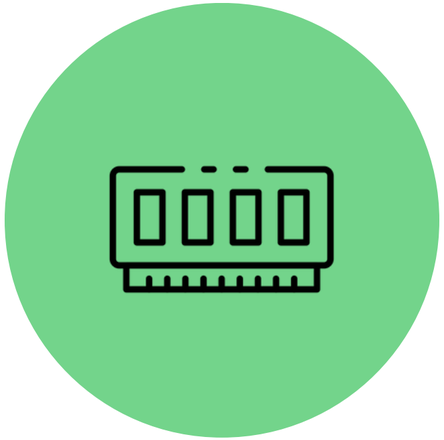
Data Structures

We created two custom data structures using typedef struct: one for Student, and another for Grade. These structures allowed us to represent and manage core entities of the system in a clear and organized way.

HOW AND WHERE?

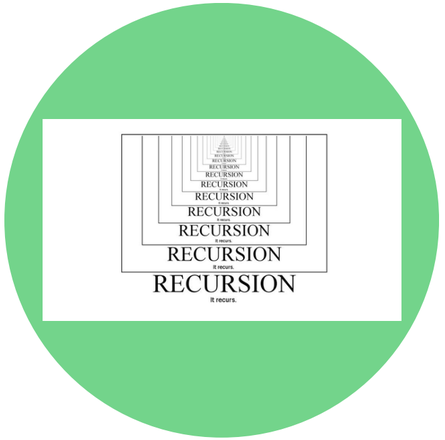
Linked List

We used linked lists to manage both students and their grades dynamically. Each student and grade entry is stored as a node in its respective linked list. This structure made it efficient to implement features.



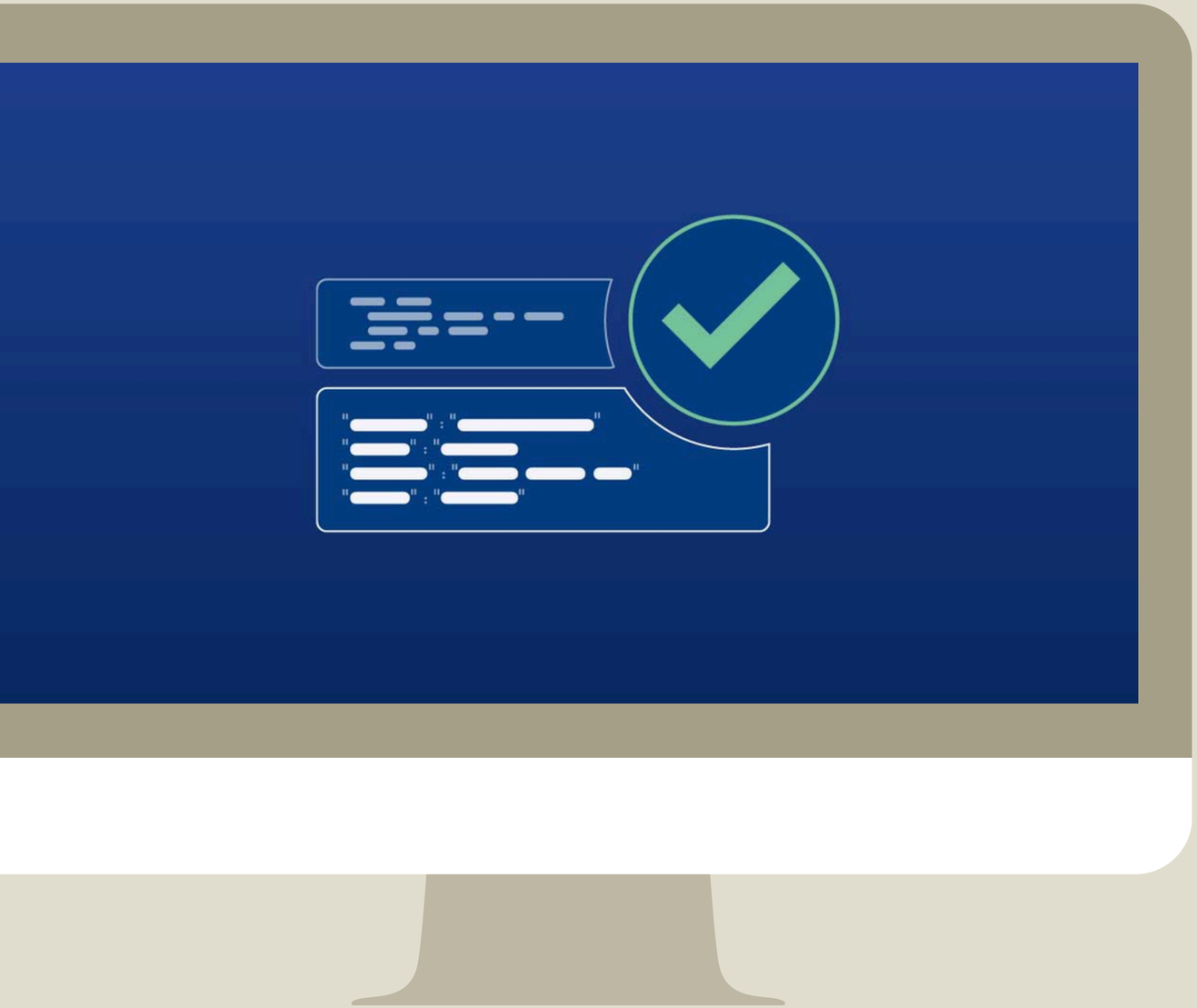
Dynamic memory allocation

Dynamic memory allocation was used to create new nodes whenever a student or grade was added. This ensured memory was only used when necessary, making the program efficient.



Recursion

At the end of the program, we used recursive functions to free all allocated memory, preventing memory leaks.



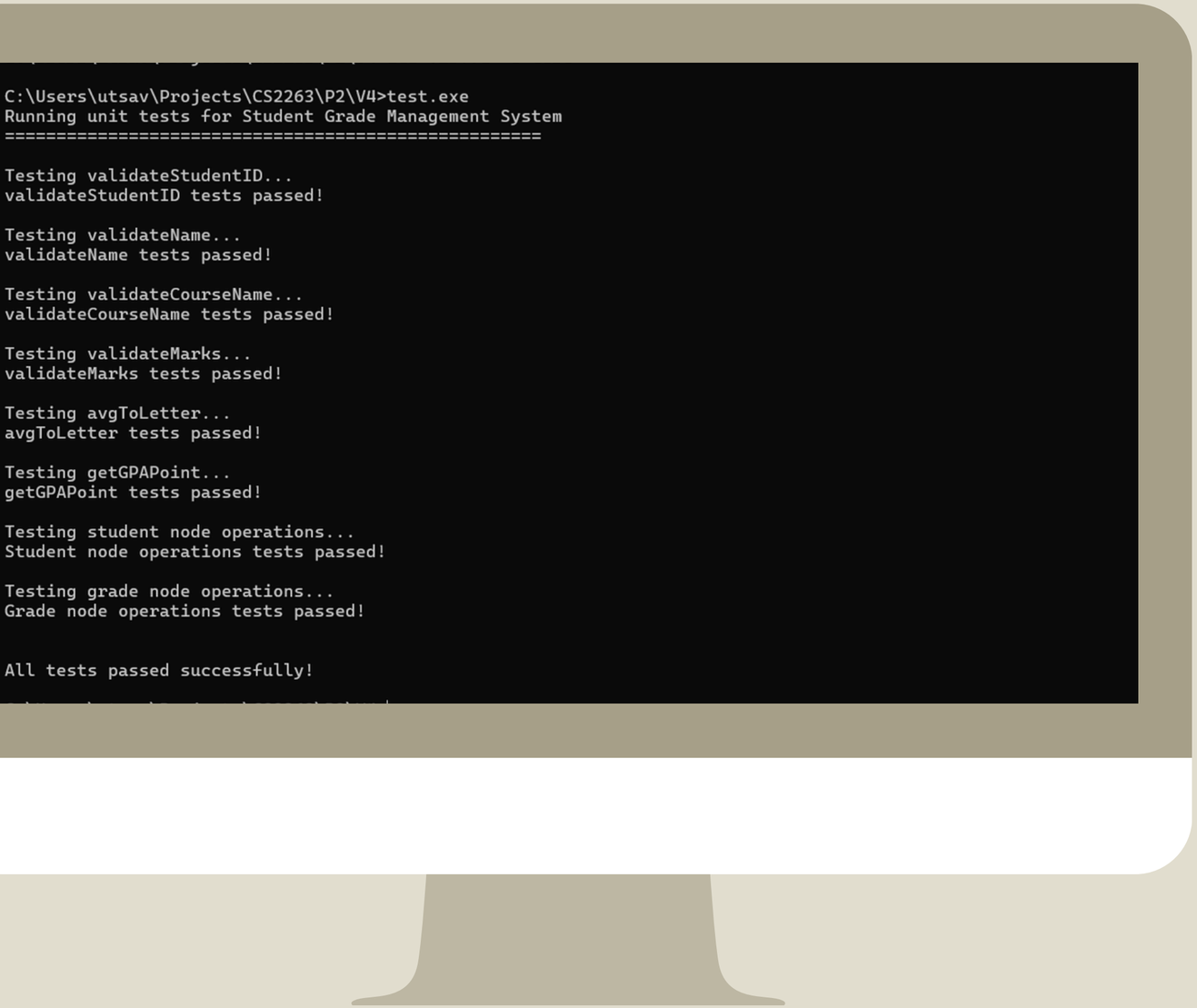
Testing and Validation

- Student ID validation (between 1000-10000)
- Name validation (alphabets & spaces)
- Course name validation (0-7 characters)
- Marks validation (between 0-100)

Error handling:

- Duplicate student ID detection
- Non-existent student ID handling
- File opening/writing error handling
- Memory allocation failure handling

by NULL checks



test.c

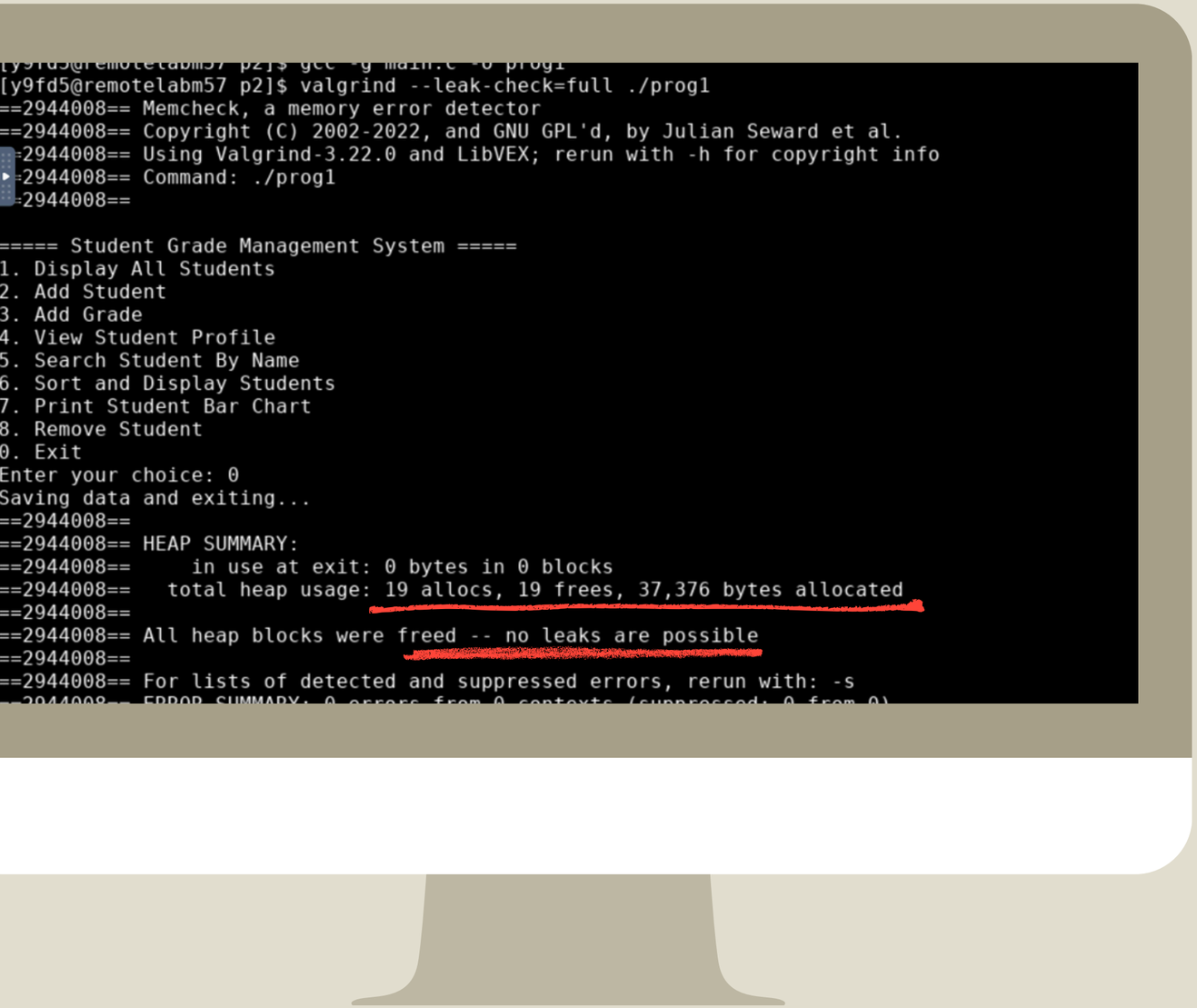
The program handles key edge cases:

- Prevents operations on empty lists
- Enforces student ID uniqueness
- Error messages to handle duplicate IDs
- Handles the full range of marks (0-100)
- Correctly assigns letter grades and GPAs
- Properly visualizes the bar chart system

All these are tested by test.c

which tends to test multiple functions, including:

- validation functions
- linkedList operations, and
- GPA calculation.



Valgrind for memory leaks

Memory Safety:

- Null checks after memory allocation
- Proper freeing of allocated memory
- Prevention of memory leaks through recursive free functions

The program was tested with Valgrind in a Linux environment.

No memory leaks were observed!

DEVELOPMENT STAGES

1	Basic Main Menu and structure definitions
2	GPA and other grade calculation functions
3	Searching and Bar chart logic along with display feature
4	Validation, tests (including edge cases) and Memory Safety
5	Linked List Implementation

TEAM CONTRIBUTIONS

Student Grade Management System

Thanh

- Main Menu
- Structure Definitions
- Student functions
- Sorting Student Functionality

Both

- Partial Student Name Search
- Linked List implementation

Utsav

- Validation functions
- Grade calculation logics
- Bar chart
- test.c

CHALLENGES

We lost a lot of time trying to fix a linking error that kept showing up.

We tried at least 10 things, but nothing worked. That's why all our code is in a single main.c file instead of being split up

CHALLENGES

TIME MANAGEMENT -

We started this project early during reading break,

but later had to switch everything to implement linked lists since that was taught near the end.

This change completely altered our program's logic, and redoing it under time pressure made us miss the deadline.

Using color in the bar chart wasn't really hard—it was actually fun to learn how C can print in color using ANSI codes. It turned out great!

CHALLENGES

LESSONS LEARNED

- Linked lists provide flexible dynamic storage
- Input validation is critical for overall data integrity
- File I/O provides simple but effective and easy to use
- ANSI color codes can significantly improve user experience in console applications

CONCLUSION

In conclusion, our Student Grade Management System successfully implements core concepts of C programming such as linked lists, dynamic memory allocation, file I/O, and custom data structures.

The program allows users to manage student records and grades efficiently with an effective menu interface.

Through testing and memory validation, we ensured the system is robust and reliable.

KNOWN LIMITATIONS

- The program only works in terminal/command-line environments
- The interface is text-based without graphical elements
- File I/O is done with text-based CSV
- Basic interface — no graphical UI

FUTURE IMPROVEMENTS

- Implement a login system for role-based access (e.g., admin, student).
- Introduce credit hours to calculate weighted GPA.
- Build a graphical user interface (GUI).
- Enhance file storage by switching to binary files

Q&A

Thank You