












COP 4338 Assignment-2: Student Grade Management System

Table of Contents

1.  [TA Contact Information](#)
2.  [Assignment Overview](#)
3.  [Skills You'll Develop](#)
4.  [Learning Objectives](#)
5.  [Assignment Tasks](#)
6.  [Project Framework](#)
7.  [Development & Testing](#)
8.  [Assignment Submission Guidelines](#)
9.  [Deliverable Folder Structure](#)
10.  [Grading Rubric](#)

Important Note!

Please ensure that your implementation works on the course-designated ocelot server: **ocelot-bbhatkal.aul.fiu.edu**. Claims such as "it works on my machine" will not be considered valid, as all testing and grading will be conducted in that environment.

 **Important:** You are provided with an autograder to support consistent testing and maintain the highest level of grading transparency. The same autograder will be used by the instructor for final evaluation.

If your implementation passes all three test cases— **simple**, **moderate**, and **rigorous**—using the autograder, you are likely to receive full credit for your submission.

 **There is only one submission - RESUBMISSION IS NOT ALLOWED.** You must test your implementation thoroughly with the provided testing framework on the Ocelot server before the final submission.

1. TA Contact Information

For any questions or concerns related to assignment grading, please reach out to the TA listed below.

Communication Guidelines:

- Ensure that all communication is polite and respectful
- You must contact the TA **first** for any grading issues. If the matter remains unresolved, feel free to reach out to me

Name: Sanskar Lohani — sloha002@fiu.edu

Sections: COP 4338-RVG (88613), COP 4338-U01 (84664)

2. Assignment Overview

This assignment provides hands-on experience with **fundamental C programming concepts, array manipulation, function implementation**. You will implement a complete **student grade management system** capable of storing student information, managing grades across multiple assessment types, calculating statistics. The system will demonstrate **structured programming principles, input validation, and data management** techniques essential for systems programming. Your implementation must exhibit **deterministic behavior** — for the same input, it must consistently produce the same output.

The system simulates a complete academic grade management environment for educational settings, requiring students to manage complex student data structures, implement comprehensive statistical calculations across multiple assessment types, handle sophisticated data operations with robust input validation, and maintain data consistency across parallel arrays—all within a structured C implementation using fundamental programming constructs including linear search algorithms, grade validation logic, and real-time statistical analysis for up to 50 students with four distinct assessment categories.

✓ Course Modules Covered:

- [Module-1](#)
- [Module-2](#)

3. Skills You'll Develop

In this assignment, you'll **gain hands-on experience** with following essential **systems programming skills** for professional software development in C.

- **Work** with parallel 1D arrays to manage related data efficiently
- **Design** and implement modular functions with proper parameters and return values
- **Handle** user input with validation and formatted output display
- **Use** loops (for, while, do-while) and conditional statements (if-else, switch)
- **Understanding** global variables scope, extern declarations, and shared data
- **Use** `#define` preprocessor directives and symbolic constants
- **Implement** robust data validation and error handling
- **Find** and retrieve data from arrays using the linear search algorithm
- **Compute** averages, statistics, and grade distributions
- **Perform** Input validation, range checking, and graceful failure management

4. Learning Objectives

- **Implement** parallel array management systems that maintain synchronized student data across multiple assessment categories.
 - **Apply** linear search algorithms and data retrieval concepts similar to database indexing and record management systems.
 - **Design** comprehensive statistical analysis logic with interdependent grade calculations and academic performance metrics.
 - **Create** robust data validation frameworks with comprehensive input checking and error prevention mechanisms.
 - **Develop** advanced mathematical computation techniques for real-time average calculations and letter grade assignments.
 - **Structure** modular function architecture using proper parameter passing and return value management for maintainable academic software.
-

5. Assignment Tasks

What You Must Implement

You will build a **complete student grade management system**.

Core Functionality:

- **Handle up to 50 students** - Respect the MAX_STUDENTS limit
 - **Validate student IDs (1-9999)** - Enforce positive integers within range
 - **Validate and store grades (0.0-100.0)** - Use MIN_GRADE and MAX_GRADE constants
 - **Prevent duplicate student IDs** - Check uniqueness before adding students (use linear/binary search)
 - **Calculate accurate averages** - Only include entered grades in calculations
 - **Display proper letter grades** - A(90+), B(80-89), C(70-79), D(60-69), F(<60)
 - **Handle "no grades" cases** - Display "N/A" when no grades are entered
 - Handles **4 assessment types**: Quiz, Assignment, Midterm, Final Exam
 - Provides **comprehensive class statistics** including min, max, average for each assessment
-

6. Project Framework

This assignment includes a comprehensive development framework comprising **skeleton code files** (*templates for your implementation*), **utility functions** (*if applicable*), a **driver program**, a **Makefile**, and an **autograder** to support and guide your implementation process.

1) Provided Framework Infrastructure (*Do Not Modify*)

- `grade_system.h`: Complete header file with data structures, constants, and function prototypes. **(study it thoroughly!)**.
- `driver.c`: The main driver program responsible for coordinating all tests and managing the overall execution flow.
- `Makefile`: Used to build your application and generate the final executable `grade_system`. You can run it using the following command:

```
1 ./grade_system
```

- `autograder_grade_system.sh`: Automated testing script **(identical to the final grading mechanism)**.
- `batchgrader_grade_system.sh`: Automated testing script **(identical to the final grading mechanism)**.

2) Student Implementation File

- `A2_skeleton.c`: Your implementation template with guided TODO sections. Rename it to → `functions.c`.

3) Testing Infrastructure

- `TESTCASES.txt`: # Test cases **(copy simple/moderate/rigorous testcases here)**.
- `EXPECTED_OUTPUT.txt`: # Expected results **(generated by executing `./A1_sample > EXPECTED_OUTPUT.txt`)**.
- **Test Case Sets**: Multiple test case sets ranging from simple to rigorous complexity. **As you progress with your implementation, copy each of these test cases into `TESTCASES.txt` one by one. You are required to pass all the test cases!**
 - `testcases_simple.txt` / `expected_output_simple.txt`: Basic validation tests.
 - `testcase_moderate.txt` / `expected_output_moderate.txt`: Intermediate complexity tests.
 - `testcase_rigorous.txt` / `expected_output_rigorous.txt`: Comprehensive stress tests.

4) Instructor's Reference Implementation

- `A2_sample`: The instructor-provided reference executable that demonstrates the **complete, correct, and required implementation** of this assignment. 💡 **This executable serves as your primary reference for understanding the expected behavior.**
- ⚠️ **Important**: Your program's output must **exactly match** that of the instructor-provided executable for equivalent input. Any deviation in behavior, logic, or output formatting may prevent the autograder from evaluating your submission.

5) Data Structure Requirements

- All data structures must use the definitions provided in `grade_system.h`. Refer to this header file **thoroughly** to understand the constants, global entities, function prototypes, and other framework components required for your implementation.

6) Provided Files Layout

```

1  A2_PROVIDED_FILES/
2  └─ Sample_Executable/
3     └─ A2_sample                # Reference implementation (study this thoroughly!)
4  └─ Framework_Files/
5     └─ grade_system.h           # Header file (provided - DO NOT MODIFY)
6     └─ driver.c                 # Main program (provided - DO NOT MODIFY)
7     └─ Makefile                 # Builds your application (provided - DO NOT MODIFY)
8     └─ autograder_grade_system.h # Autograder (provided - DO NOT MODIFY)
9     └─ batchgrader_grade_system.h # Batch autograder (provided - DO NOT MODIFY)
10    └─ TESTCASES.txt            # You will copy-paste the testcases here
11    └─ EXPECTED_OUTPUT.txt      # You will copy-paste the expected output here
12    └─ A2_skeleton.c            # Your implementation file (rename to functions.c for
    submission)
13  └─ Testcases/
14     └─ testcases_simple.txt     # Simple testcases
15     └─ testcase_moderate.txt   # Moderate testcases
16     └─ testcase_rigorous.txt   # Rigorous testcase
17  └─ A2_Specification.pdf        # This detailed specification & requirement document
18  └─ A2_Testing_Guide           # A comprehensive testing guide
19  └─ README.txt                 # You must submit team details in the same format

```

7) Execution Permissions Notice

If you **do not** have execute permissions for instructor-provided framework files, you can manually assign the necessary permissions to specific files or to all files within a directory using the following command on a **Linux system**:

Note: When files are uploaded or transferred to Ocelot server (`ocelot-bbhatka1.aul.fiu.edu`), execute permissions **may be stripped** due to security restrictions. In such cases, you must explicitly grant **read (r)**, **write (w)**, and **execute (x)** permissions.

[Granting Permissions to All Items in the Current Directory](#)

```

1  # Check current permissions
2  ls -l
3
4  # Grant read, write, and execute permissions (safe and preferred - applies only what's missing)
5  chmod +rwx *
6
7  # Assign full permissions (read, write, execute) to all users for all files in the current
  directory
8  # ⚠ Not preferred - use only as a last resort!
9  chmod 777 *
10
11 # Verify that permissions were applied
12 ls -l

```

 **Caution: Use `chmod 777` with care!**

Granting full permissions to everyone can:

- Expose sensitive data

- Allow unintended modifications or deletions
- Introduce security risks, especially in shared or multi-user environments

8) YOU MUST!


- **Test your implementation thoroughly** on the **Ocelot** server (`ocelot-bbhatka1.aul.fiu.edu`)
- **Ensure exact output matching** with the instructor sample executable `A2_sample`
- **Validate all calculations** and logic against the sample
- **Test edge cases** and error handling extensively
- **Verify compilation** with the specified compiler flags

7. Development & Testing


Required Files for Development

All the following files must be in the same directory:

1. **Framework files:** `driver.c`, `Makefile`, `grade_system.h`
2. **Your implementation files:** `functions.c`
3. **Testing tools:** `autograder_grade_system.sh`, `batchgrader_grade_system.sh`, `TESTCASES.txt`, `EXPECTED_OUTPUT.txt`


 **Critical Testing Information:** The instructor will use different and more rigorous test cases for final grading that are not provided to students. However, if you pass all provided rigorous test cases with 100% accuracy, you are highly likely to pass the instructor's final grading test cases..

Comprehensive Development Workflow

 **Important:** Please refer to the document [A2_Testing_Guide.pdf](#) for the detailed testing instructions.

Step 1: Understand Expected Output Format

 The file `TESTCASES.txt` must be present in the directory where `A2_sample` is located.

 The file `TESTCASES.txt` should contain the test cases you wish to run to generate the `EXPECTED_OUTPUT.txt`.

```
1 # Generate expected output from instructor sample
2 ./A2_sample > EXPECTED_OUTPUT.txt
3
4 # View the expected output format to understand correct behavior
5 cat EXPECTED_OUTPUT.txt
```

Step 2: Implement with Incremental Testing

- **Test with Progressive Complexity:**
 - ☒ Please refer to the document [A2_Testing_Guide.pdf](#) for the detailed progressive testing instructions.

Step 3: Use Autograders for Final Validation

-  **Important:** Please note that instructor will use the same autograders for the final grading with different testcases.




Autograder System



The autograder compares your program's output with pre-provided reference output to determine correctness.

Internal Process:

1. **Clean rebuild:** `make clean && make`
2. **Runs your implementation:** `./grade_system > STUDENT_OUTPUT.txt`
3. **Compares outputs:** Extracts each data structure section and compares with expected results
4. **Calculates scores:** Awards points based on percentage of matching output lines
5. **Provides detailed feedback:** Shows exactly which lines don't match

8. Assignment Submission Guidelines

- All assignments **must be submitted via Canvas** by the specified deadline as a **single compressed (.ZIP) file**. Submissions by any other means **WILL NOT** be accepted and graded!
- The filename must follow the format: `A2_FirstName_LastName.zip` (e.g., [A1_Harry_Potter.zip](#))
-  **Important:** Include the Firstname and Lastname of the **team's submitting member**, or your own name if **submitting solo**.
-  **Important:** For **group submissions**, each group must designate one **corresponding member**. This member will be responsible for submitting the assignment and will serve as the primary point of contact for all communications related to that assignment.
-  **Important Policy on Teamwork and Grading**

All members of a team will receive the **same grade** for collaborative assignments. It is the responsibility of the team to ensure that work is **distributed equitably** among members. Any disagreements or conflicts should be addressed and resolved within the team in a **professional manner**. [If a resolution cannot be reached internally, the instructor may be requested to intervene.](#)
-  **Important:** Your implementation **must be compiled and tested on the course designated Ocelot server** (`ocelot-bbhatka1.aul.fiu.edu`) **to ensure correctness and compatibility**. [Submissions that fail to compile or run as expected on Ocelot will receive a grade of ZERO.](#)
- Whether submitting as a **group or individually**, every submission must include a **README** file containing the following information:
 1. Full name(s) of all group member(s)
 2. **Prefix the name of the corresponding member with an asterisk** 
 3. Panther ID (PID) of each member

4. FIU email address of each member
 5. Include any relevant notes about compilation, execution, or the programming environment needed to test your assignment. **Please keep it simple—avoid complex setup or testing requirements.**
- **⚠ Important - Late Submission Policy:** For each day the submission is delayed beyond the **Due date** and until the **Until date** (as listed on Canvas), a **10% penalty** will be applied to the total score. **Submissions after the Until date will not be accepted or considered for grading.**
 - **Due Date:** ANNOUNCED ON CANVAS
 - **Until Date:** ANNOUNCED ON CANVAS
 - Please refer to the course syllabus for any other details.

9. Deliverable Folder (ZIP file) Structure - ⚠ exact structure required - no additional files/subfolders

⚠ **DO NOT submit any other files other than these required files. The batch autograder may treat any additional items in the ZIP file as invalid, which will result in a grade of zero: functions.c , README.txt**

📁 All other required framework files and the testcases will be supplied by the instructor to the autograder during final grading.

- ⚠ You are required to submit your work as **a single ZIP archive file**.
- ⚠ **Filename Format:** your `Firstname_Lastname.zip` (exact format required).
- **You will be held responsible for receiving a ZERO grade if the submission guidelines are not followed.**

```
1 A2_FirstName_LastName.zip
2 |— functions.c      # Your complete implementation for 8 functions
3 |— README.txt      # Team details (see submission guidelines)
```

10. ¹⁰⁰ Grading Rubric

Criteria	Marks
⚠ Program fails to run or crashes during testing	0
⚠ Partial Implementation (<i>Evaluated based on completeness and at the instructor's discretion</i>)	0 - 70

Detailed Breakdown of 100 Points

Function	Points
<code>isValidGrade</code> implementation	5
<code>getLetterGrade</code> implementation	5
<code>findStudentByID</code> implementation	10

Function	Points
<code>calculateStudentAverage</code> implementation	10
<code>addStudent</code> implementation	10
<code>enterGrade</code> implementation	10
<code>displayStudentGrades</code> implementation	10
<code>calculateStatistics</code> implementation	30
TOTAL	90
The final 10 points	Code quality, Proper comments, Clean structure, Adherence to guidelines

Good luck! 🍀