ICS 365. Organization of Programming Languages

Programming Assignment 2 - Matrix Multiplication

Due: *October 8, 2025 at 6:00 PM*
Points: *100*

Overview

You will build a C program that:

1. Prompts the user for the dimensions of two matrices.
2. Validates whether multiplication is possible.
3. Uses **dynamic memory allocation** to store the matrices.
4. Performs matrix multiplication.
5. Prints the resulting matrix in a formatted manner.

This assignment practices:

- **C input/output**
- **Dynamic memory allocation (malloc, free)**
- **Functions**
- **Matrix operations**

Program Behavior

1. Ask the user for the dimensions of both matrices.
    - For Matrix A: `rowA colA`

- For Matrix B: `rowB colB`

2. Validate dimensions:
   - If `colA != rowB`, multiplication is not possible.
   - Print an error and terminate.
3. Allocate memory dynamically for matrices A, B, and C.
4. Prompt the user to enter elements for both matrices.
5. Perform multiplication using the provided pseudocode.
6. Print the resulting matrix.

Pseudocode

```text
function matrix_multiply(A, B, C, rowA, colA, colB)
    for i = 0 to rowA - 1
        for j = 0 to colB - 1
            C[i][j] = 0
            for k = 0 to colA - 1
                C[i][j] = C[i][j] + (A[i][k] * B[k][j])
            end for
        end for
    end for
end function
```

Program Specification

1. Implement the following functions in addition to `main`:
   - `create_empty_matrix`: Prompts the user for rows and columns, allocates memory dynamically for an empty matrix, and returns a pointer to the allocated matrix.
   - `read_matrix`: Reads values into a matrix from user input.

- `matrix_multiply` : Performs multiplication given dimensions.
- `print_matrix` : Displays the matrix in row/column format.

2. **Input Validation**
   - Dimensions must be positive integers (rows > 0, cols > 0).
   - Dimensions must be within a reasonable upper bound (e.g., `MAX_DIM = 1000`) to prevent huge allocations.
   - Always check the return value of `scanf` when reading input; if invalid, print an error and exit.
   - If the number of columns of the first matrix does not match the number of rows of the second, print:

     ```
     Error: Incompatible dimensions for multiplication.
     ```

     and exit.
3. **Memory Management**
   - Use `malloc` to allocate matrices.
   - Use `free` to release allocated memory at the end.
   - Proper memory management is expected: allocate exactly what is required, check for allocation success, and free all allocated memory when done.

Sample Run

```sh
Enter rows and columns of first matrix: 2 3
Enter rows and columns of second matrix: 3 2

Enter elements of first matrix: 1 2 3 4 5 6
Enter elements of second matrix: 7 8 9 10 11 12

Result:
58 64
139 154
```

File Layout (Multi-file)

```
/project-root
  ├── include/
  │   └── matrix_utils.h
  ├── src/
  │   ├── main.c
  │   └── matrix_utils.c
  ├── README.md
  └── reflection.md
```

Constraints

- Use **dynamic memory allocation** (`malloc` , `free`).
- Matrices must be passed to functions as pointers.
- Functions should be modular and well-documented (use **C-style documentation comments**).
- Proper memory management is required — allocate only what is necessary and free all memory after use.

Deliverables

- Source code (`main.c` , `matrix_utils.c` , `matrix_utils.h`).
- `README.md` with compile and run instructions.
- `reflection.md` per syllabus requirements.

Evaluation (100 points)

- **Correctness (60 pts)**: Program correctly performs multiplication and handles validation.
- **Spec compliance (20 pts)**: Meets requirements and constraints.
- **Code quality (12 pts)**: Uses clear naming, formatting, modular functions, and proper documentation.
- **User experience (8 pts)**: Clear prompts, error handling, and formatted output.

Program Submission Instructions

Submission Process

- Push your code to the **GitHub repository** provided.
- Repo name format:

  ```
  ics365-fall-2025-firstname_lastname
  ```

- Submit the GitHub repository link in D2L (tracking only).
- Only push code in the specified directory.

Commit & Deadline Policy

- Commits **after the deadline will not be evaluated**.
- Repository will be reverted to the **last commit before the deadline**.
- Single late commit = no submission.
- Gradual, meaningful commits are expected.

Required Files

- **README.md**: How to compile and run.
- **reflection.md**: As described in syllabus.
- Source files and headers.