Project Title: Universal Converter Library in C

1. Introduction

This document outlines the requirements for developing a comprehensive "Universal Converter Library" in the C programming language. The goal of this project is to create a robust and modular library capable of performing various number base conversions and common unit conversions. Students will be provided with existing C source files (converter.c, main.c, converter.h) and a Makefile as a starting point. Your task is to understand, potentially enhance, and ensure the correct functionality of this provided codebase.

This project will reinforce your understanding of C programming concepts, including:

- Function design and implementation
- Header files and modular programming
- String manipulation
- Basic arithmetic and data types
- Compilation using Makefiles

2. Project Objectives

Upon successful completion of this project, students will be able to:

- Demonstrate proficiency in C programming for utility development.
- Understand and implement functions for number base conversions.
- Understand and implement functions for various unit conversions.
- Apply input validation techniques to ensure robust program behavior.
- Utilize Makefiles for efficient project compilation and management.
- Perform basic testing and verification of C code.

3. Functional Requirements

The converter library must provide the following functionalities:

3.1. Number Base Conversions

The library shall support conversions between the following number bases:

- **Binary to Decimal:** Convert a binary string representation to its decimal integer equivalent.
- **Octal to Decimal:** Convert an octal string representation to its decimal integer equivalent.
- **Hexadecimal to Decimal:** Convert a hexadecimal string representation to its decimal integer equivalent.
- **Decimal to Binary:** Convert a decimal integer to its binary string representation.
- **Decimal to Octal:** Convert a decimal integer to its octal string representation.
- **Decimal to Hexadecimal:** Convert a decimal integer to its hexadecimal string representation.

3.2. Unit Conversions

The library shall support conversions for the following unit categories:

3.2.1. Length Conversion

- Supported units: meters (m), centimeters (cm), millimeters (mm).
- Conversions should be possible between any two of these units (e.g., m to cm, mm to m).

3.2.2. Weight Conversion

- Supported units: kilograms (kg), grams (g), milligrams (mg).
- Conversions should be possible between any two of these units (e.g., kg to g, mg to kg).

3.2.3. Temperature Conversion

- Supported units: Celsius (C), Fahrenheit (F), Kelvin (K).
- Conversions should be possible between:
    - Celsius and Fahrenheit
    - Celsius and Kelvin
    - Fahrenheit and Celsius
    - Kelvin and Celsius

3.3. Input Validation

The library shall include functions to validate input for:

- **Base Conversion Input:** Verify if a given string is a valid representation for a specified base (binary, octal, hexadecimal).
- **Unit Input:** Verify if a given unit string is valid for a specific conversion category (length, weight, temperature).

4. Technical Requirements and Constraints

- **Programming Language:** All code must be written in C (C11 standard).
- **Modularity:** The project must be structured into separate source (.c) and header (.h) files as provided (converter.c, main.c, converter.h).
- **Compilation:** The project must compile successfully using the provided Makefile and gcc compiler.
- **Standard Libraries:** Only standard C libraries (stdio.h, stdlib.h, string.h, math.h, ctype.h) are permitted.
- **No Global Variables:** Avoid the use of global variables for state management within the converter.c functions. All necessary data should be passed via function parameters.
- **Error Handling:** Functions should return appropriate values (e.g., -1.0 for invalid conversions, 0/1 for validation) to indicate success or failure where applicable.

5. Deliverables

Students are required to submit the following files:

- converter.c: The implementation of the conversion and validation functions.
- converter.h: The header file declaring the functions.
- main.c: The test program demonstrating the functionality of the library.
- Makefile: The build script for compiling the project.

6. Testing Guidelines

A main.c file has been provided as a test suite. Students should:

1. **Understand the Test Cases:** Analyze main.c to understand how each function in converter.c is called and what the expected outputs are.
2. **Verify Outputs:** Compile and run main.c to ensure that all actual outputs match the expected outputs.
3. **Edge Cases (Optional but Recommended):** Consider adding additional test cases to main.c to cover edge cases or unusual inputs (e.g., very large numbers, zero values, invalid unit combinations not explicitly caught by is_valid_unit).

7. Submission Guidelines

- All source code files (.c, .h, Makefile) should be submitted together in a single archive (e.g., .zip or .tar.gz).
- Ensure your code is well-commented, especially for complex logic or non-obvious implementations.
- The code should be clean, readable, and adhere to good C programming practices.

8. Evaluation Criteria

Projects will be evaluated based on:

- **Correctness (60%):** All conversion and validation functions produce accurate results for various inputs, as demonstrated by main.c and potentially additional test cases.
- **Code Quality (20%):** Readability, adherence to C standards, proper use of modularity, and effective commenting.
- **Makefile Functionality (10%):** The Makefile correctly compiles the project without errors or warnings.
- **Adherence to Requirements (10%):** All specified functional and technical requirements are met.