

Bài tập 1: Trong các ứng dụng thực tế, việc chỉ dùng printf để gõ lỗi và ghi lại thông tin là không đủ. Một hệ thống logger chuyên nghiệp cần có khả năng:

1. Ghi log ra nhiều nơi khác nhau (console, file).
2. Lọc các thông điệp log dựa trên mức độ nghiêm trọng (ví dụ: chỉ hiển thị lỗi, bỏ qua thông tin gỡ lỗi).
3. Tự động thêm các thông tin hữu ích như timestamp, tên file, và số dòng vào mỗi thông điệp log.

Bài tập này yêu cầu bạn xây dựng một module logger hoàn chỉnh, có thể được cấu hình và sử dụng trong bất kỳ dự án C nào, tuân thủ các quy tắc về mã sạch và tổ chức module.

Yêu cầu phần mềm:

1. Cấu trúc mã nguồn
 - a. logger.h
 - b. logger.c
 - c. main.c
2. Các mức Log (Log Levels): Sử dụng enum để định nghĩa 8 cấp độ log theo tiêu chuẩn của syslog:
 - a. LOG_EMERGENCY
 - b. LOG_ALERT
 - c. LOG_CRITICAL
 - d. LOG_ERROR
 - e. LOG_WARNING
 - f. LOG_NOTICE
 - g. LOG_INFO
 - h. LOG_DEBUG.
3. Định dạng Log
 - a. Mỗi dòng log được ghi ra phải có định dạng chuẩn sau:
 - i. [YYYY-MM-DD HH:MM:SS] [LEVEL] [FILENAME:LINE] - Message
 - ii. Ví dụ: [2024-07-20 11:45:10] [ERROR] [main.c:25] - Failed to connect to database.

Yêu cầu chức năng:

1. **Logger đa đầu ra (Multi-target Logger)**
 - a. Logger phải có khả năng ghi log đồng thời ra màn hình console (stderr cho các lỗi nghiêm trọng và stdout cho các log thông thường) và một file văn bản.
 - b. Việc ghi vào file là tùy chọn và được cấu hình khi khởi tạo.

2. Lọc theo cấp độ (Level Filtering)

- a. Logger phải có thể được cấu hình với một mức log tối thiểu (ví dụ: LOG_INFO).
- b. Bất kỳ thông điệp nào có mức độ ưu tiên thấp hơn (giá trị số lớn hơn) mức đã cấu hình sẽ bị bỏ qua và không được ghi lại.
- c. Cần có một hàm để thay đổi mức lọc này tại thời điểm chạy.

3. Tự động thêm Metadata

- a. Logger phải tự động lấy và chèn timestamp, tên file nguồn, và số dòng vào đầu mỗi thông điệp log.
- b. Gợi ý: Sử dụng các macro có sẵn của trình biên dịch là __FILE__ và __LINE__.

4. Giao diện tiện lợi (Convenience Interface)

- a. Tạo một macro log_message(...) để người dùng có thể gọi một cách thuận tiện mà không cần phải tự điền __FILE__ và __LINE__.

Mục tiêu cần đạt được

1. Nắm vững cách sử dụng hàm với danh sách đối số thay đổi (stdarg.h).
2. Sử dụng thành thạo snprintf và vsnprintf để định dạng chuỗi an toàn.
3. Hiểu và áp dụng kỹ thuật I/O file (fopen, fprintf, fclose, fflush).
4. Sử dụng static để đóng gói trạng thái trong một module C.
5. Tạo và sử dụng các macro hữu ích (__FILE__, __LINE__).
6. Xây dựng một module C hoàn chỉnh, có tính tái sử dụng cao.

Lưu ý: Không dùng Design Pattern.