Dưới đây là ví dụ hoàn chỉnh về **một đoạn code C bị lỗi Segmentation fault**, kèm **2 prompt khác nhau** và **phân tích sự khác biệt rõ ràng giữa 2 kết quả**.

**1. Đoạn code bị lỗi**

#include <stdio.h>

#include <string.h>

int main() {

char \*str = "Hello";

str[0] = 'h'; // Thay đổi ký tự đầu

printf("%s\n", str);

return 0;

}

**Lỗi:**

Chương trình sẽ **bị Segmentation fault** khi chạy.

**2. Prompt 1: “Sửa lỗi chương trình trên”**

**Kết quả AI trả về (ngắn gọn, chỉ tập trung vào việc sửa lỗi)**

#include <stdio.h>

#include <string.h>

int main() {

char str[] = "Hello"; // Dùng mảng ký tự thay vì con trỏ hằng

str[0] = 'h';

printf("%s\n", str);

return 0;

}

**Giải thích ngắn (nếu có):**

Dùng char[] để tạo chuỗi có thể sửa đổi, vì chuỗi hằng char \* không được phép thay đổi.

**3. Prompt 2:**

“Hãy giải thích chi tiết tại sao xảy ra lỗi Segmentation fault, minh họa bằng sơ đồ bộ nhớ, sau đó sửa lỗi.”

**Kết quả AI trả về (chi tiết hơn, có phân tích và sơ đồ)**

**Giải thích nguyên nhân:**

Lỗi xảy ra vì dòng:

char \*str = "Hello";

trỏ tới một **chuỗi hằng (string literal)**, vùng nhớ này nằm trong **phân vùng “read-only” của bộ nhớ chương trình**.  
Khi ta cố gắng thay đổi str[0] = 'h';, chương trình truy cập vùng nhớ chỉ đọc → gây lỗi **Segmentation fault**.

**Minh họa sơ đồ bộ nhớ:**

+-----------------------+

| Code segment |

+-----------------------+

| Read-only data ("Hello") <--- str trỏ đến đây

| H e l l o \0 |

| ↑ |

| str[0] = 'h' | (không được phép ghi)

+-----------------------+

| Stack (biến cục bộ) |

| Heap (malloc, v.v.) |

+-----------------------+

**Cách sửa lỗi đúng:**

#include <stdio.h>

int main() {

char str[] = "Hello"; // Tạo bản sao trong stack (ghi được)

str[0] = 'h'; // Hợp lệ

printf("%s\n", str);

return 0;

}

**Giải thích thêm:**

* Dòng char str[] = "Hello"; tạo một **mảng ký tự có thể ghi** trên stack.
* Việc sửa str[0] lúc này chỉ thao tác trong vùng nhớ hợp lệ → không lỗi.

**4. So sánh hai cách trả lời**

| **Tiêu chí** | **Prompt 1: “Sửa lỗi”** | **Prompt 2: “Giải thích nguyên nhân + sửa lỗi”** | **Khác biệt / Lợi ích** |
| --- | --- | --- | --- |
| **Mục tiêu trả lời** | Chỉ tập trung đưa ra code đúng | Phân tích nguyên nhân, mô phỏng bộ nhớ, rồi sửa lỗi | Prompt 2 giúp hiểu bản chất lỗi |
| **Độ chi tiết** | Ngắn gọn, chỉ vài dòng | Phân tích kỹ, có sơ đồ, diễn giải cơ chế bộ nhớ | Hiểu sâu hơn về “vì sao lỗi xảy ra” |
| **Khả năng học hỏi** | Người dùng chỉ thấy cách làm đúng | Người dùng hiểu **cơ chế gây lỗi + cách tránh về sau** | Prompt 2 có giá trị học tập cao |
| **Phù hợp với ai** | Khi chỉ cần chạy được | Khi muốn hiểu rõ và rèn kỹ năng debug | Prompt chi tiết hữu ích cho sinh viên, người học lập trình |

**5. Nhận xét tổng quan**

* **Prompt ngắn ("Sửa lỗi")** khiến AI chỉ tập trung vào **đưa ra kết quả đúng**, bỏ qua việc giải thích lý do.
* **Prompt chi tiết (“Giải thích nguyên nhân + minh họa sơ đồ bộ nhớ”)** buộc AI phải:
  + Phân tích cơ chế bộ nhớ (vì sao lỗi xảy ra),
  + Minh họa trực quan vùng nhớ,
  + Sau đó mới đưa ra giải pháp khắc phục.

=> Prompt càng rõ ràng và cụ thể về **mục đích học** (không chỉ “đúng code”) thì câu trả lời càng **giàu thông tin, mang tính giáo dục** hơn.