Dưới đây là **một bài thực hành hoàn chỉnh** cho sinh viên theo yêu cầu của bạn:

Chọn **hàm kiểm tra số nguyên tố trong C** làm ví dụ, sau đó dùng **2 prompt khác nhau** để quan sát sự khác biệt giữa câu trả lời của AI.

**1. Chọn bài toán**

Viết hàm C kiểm tra một số nguyên dương n có phải là số nguyên tố hay không.

**2. Prompt 1 (ngắn, trực tiếp)**

Viết hàm C kiểm tra số nguyên tố.

**Kết quả AI sinh ra (code ngắn, cơ bản)**

#include <stdio.h>

int isPrime(int n) {

if (n < 2) return 0;

for (int i = 2; i \* i <= n; i++) {

if (n % i == 0)

return 0;

}

return 1;

}

int main() {

int n;

scanf("%d", &n);

if (isPrime(n))

printf("So nguyen to");

else

printf("Khong phai so nguyen to");

return 0;

}

**Đặc điểm:**

* Ngắn gọn, chạy đúng.
* Không có chú thích.
* Biến đặt tên cơ bản, dễ hiểu nhưng không mô tả chi tiết.
* Không có hướng dẫn nhập/xuất cho người dùng.

**3. Prompt 2 (đầy đủ, mô tả rõ ràng)**

Viết hàm C kiểm tra số nguyên tố, **có chú thích từng bước**, **đầu vào là số nguyên dương n**, **đặt tên biến rõ ràng**, và **in kết quả dễ hiểu cho người dùng**.

**Kết quả AI sinh ra (chi tiết, có chú thích)**

#include <stdio.h>

#include <stdbool.h>

// Hàm kiểm tra số nguyên tố

bool laSoNguyenTo(int n) {

// Số nhỏ hơn 2 không phải số nguyên tố

if (n < 2)

return false;

// Kiểm tra từ 2 đến căn bậc hai của n

for (int i = 2; i \* i <= n; i++) {

// Nếu chia hết, n không phải số nguyên tố

if (n % i == 0)

return false;

}

// Nếu không chia hết cho bất kỳ số nào → là số nguyên tố

return true;

}

int main() {

int n;

// Nhập dữ liệu

printf("Nhap mot so nguyen duong: ");

scanf("%d", &n);

// Kiểm tra và in kết quả

if (laSoNguyenTo(n))

printf("%d la so nguyen to.\n", n);

else

printf("%d khong phai la so nguyen to.\n", n);

return 0;

}

**Đặc điểm:**

* Có **chú thích chi tiết từng bước** trong hàm.
* Dùng **kiểu bool** để biểu diễn đúng logic.
* Biến và hàm đặt tên **rõ ràng, có ý nghĩa tiếng Việt**.
* Có hướng dẫn nhập/xuất rõ ràng cho người dùng.
* Code dài hơn, trình bày chuẩn và dễ đọc.

**4. So sánh kết quả sinh ra từ hai prompt**

| **Tiêu chí** | **Prompt 1 (ngắn)** | **Prompt 2 (dài, chi tiết)** | **Nhận xét / Cải thiện** |
| --- | --- | --- | --- |
| **Độ dài prompt** | Rất ngắn, chỉ nêu nhiệm vụ | Mô tả rõ yêu cầu (chú thích, tên biến, đầu vào, cách in) | Prompt chi tiết giúp AI hiểu rõ mục tiêu hơn |
| **Cấu trúc code** | Đơn giản, đủ chạy | Có cấu trúc tốt, dễ bảo trì | Prompt chi tiết hướng AI đến code “chuẩn mực” |
| **Chú thích** | Không có | Có chú thích giải thích từng phần | Giúp người học dễ hiểu luồng logic |
| **Đặt tên biến/hàm** | isPrime, n | laSoNguyenTo, gioiHan, so, v.v. | Dễ đọc, diễn đạt rõ nghĩa |
| **Đầu vào / đầu ra** | Không hướng dẫn nhập | Có lời nhắc “Nhap mot so nguyen duong” | Giao diện người dùng thân thiện hơn |
| **Mức độ học được từ kết quả** | Chỉ thấy cách viết hàm cơ bản | Hiểu cả **cách hoạt động**, **cách trình bày**, **phong cách lập trình** | Prompt chi tiết tăng tính học thuật |

**5. Kết luận**

* Prompt **ngắn gọn** (“Viết hàm kiểm tra số nguyên tố”) giúp AI tạo code nhanh, phù hợp khi bạn **chỉ cần kết quả chạy được**.
* Prompt **chi tiết** (“... có chú thích từng bước, đặt tên rõ ràng, đầu vào n...”) giúp AI sinh ra code **dễ hiểu, có tính hướng dẫn**, phù hợp cho **sinh viên đang học lập trình**.

=>> **Kết luận tổng quát:**  
Cách diễn đạt prompt **ảnh hưởng trực tiếp đến độ sâu của câu trả lời**.  
Prompt càng cụ thể, AI càng sinh ra **kết quả đúng mục tiêu học tập hơn**, không chỉ “chạy được” mà còn “hiểu được”.