### **Đề Bài: Mở Rộng và Tùy Chỉnh Hệ Thống Xử Lý Lỗi trong Lập Trình C**

#### Mục Đích Bài Tập

Mục tiêu của bài tập này là để bạn hiểu rõ hơn về việc sử dụng macro trong lập trình C, cũng như cách thiết lập một hệ thống xử lý lỗi linh hoạt và hiệu quả. Bạn sẽ sử dụng đoạn code đã cung cấp như là cơ sở để mở rộng và tùy chỉnh.

#### Nhiệm Vụ Cụ Thể

* Hiểu và Phân Tích Code Đã Cho:
  + Nghiên cứu đoạn code mẫu đã cung cấp. Hiểu rõ về cách các hàm và macro được sử dụng để xử lý các tình huống lỗi khác nhau.
* Thêm Hành Động Xử Lý Lỗi:
  + Bổ sung thêm ít nhất hai hành động xử lý lỗi mới vào hệ thống.
  + Ví dụ:

| // Hành động 1: In thông điệp lỗi void printError(const char \*message) {  printf("Error: %s\n", message); } // Hành động 2: In thông điệp cảnh báo void printWarning(const char \*message) {  printf("Warning: %s\n", message); } |
| --- |

* Tùy Chỉnh Macro **HANDLE\_ERROR**:
  + Sửa đổi macro HANDLE\_ERROR để nó có thể hỗ trợ các loại hành động xử lý lỗi mới mà bạn đã thêm vào.
  + Đảm bảo rằng các sửa đổi của bạn không làm mất đi tính linh hoạt và đa dạng của macro.
  + Ví dụ:

| HANDLE\_ERROR(x == 10, printError, "x is not equal to 10"); HANDLE\_ERROR(x < 10, printWarning, "x is less than 10"); |
| --- |

Code mẫu:

| #include <stdio.h>  // Hành động 1: In thông điệp lỗi void printError(const char \*message) {  printf("Error: %s\n", message); }  // Hành động 2: In thông điệp cảnh báo void printWarning(const char \*message) {  printf("Warning: %s\n", message); }  //Bổ sung nội dung vào  // Hành động 3: In thông điệp thông tin void printInfo(const char \*message) { }  // Hành động 4: In thông điệp gỡ lỗi void printDebug(const char \*message) { }  // Hành động 5: In thông điệp khẩn cấp void printCritical(const char \*message) { }  // Hành động 6: In thông điệp thành công void printSuccess(const char \*message) { }   // Macro xử lý lỗi #define HANDLE\_ERROR(condition, action, message) do { \  //Bổ sung nội dung chương trình vào đây  } while(0)  int main() {  int x = 5;   // Sử dụng HANDLE\_ERROR với các hành động khác nhau  HANDLE\_ERROR(x == 10, printError, "x is not equal to 10");  HANDLE\_ERROR(x < 10, printWarning, "x is less than 10");  return 0; } |
| --- |

Kết quả:

| Error: x is not equal to 10 |
| --- |

**Đề bài: Tạo hai file: một file tiêu đề (header) và một file nguồn (source)**.

File tiêu đề sẽ chứa khai báo của một hàm, và sử dụng các macro #ifndef, #define, và #endif để ngăn chặn việc bao gồm nó nhiều lần. File nguồn sẽ định nghĩa hàm đó và sử dụng hàm trong hàm main.

1. Tạo File Tiêu Đề (Header File): math\_utils.h

| // Kiểm tra xem macro MATH\_UTILS\_H đã được định nghĩa hay chưa #ifndef MATH\_UTILS\_H #define MATH\_UTILS\_H  // Khai báo hàm int add(int a, int b);  #endif // Kết thúc của #ifndef |
| --- |

Trong đoạn code trên, #ifndef MATH\_UTILS\_H kiểm tra xem MATH\_UTILS\_H đã được định nghĩa trước đó hay chưa. Nếu chưa, #define MATH\_UTILS\_H sẽ được thực thi, ngăn chặn việc bao gồm file này nhiều lần.

1. Tạo File Nguồn (Source File): main.c

| #include <stdio.h> #include "math\_utils.h" // Bao gồm file tiêu đề  // Định nghĩa hàm đã khai báo trong file tiêu đề int add(int a, int b) {  return a + b; }  int main() {  printf("Sum of 3 and 5 is %d\n", add(3, 5));  return 0; } |
| --- |

Trong file main.c, hàm add được định nghĩa, và sau đó được gọi trong hàm main.

Giải thích được tại sao phải sử dụng #ifndef, #define, và #endif