# **Quy Tắc Cấu Trúc Code C++ Theo Tiêu Chuẩn AUTOSAR Adaptive**

## **1. Giới thiệu**

AUTOSAR Adaptive là nền tảng cho các hệ thống nhúng hiện đại, đặc biệt là trong lĩnh vực ô tô. Phát triển phần mềm trên nền tảng này đòi hỏi phải tuân thủ các quy tắc lập trình nghiêm ngặt để đảm bảo tính bảo trì, mở rộng, và an toàn của phần mềm. Việc tuân thủ các tiêu chuẩn và quy tắc khi viết mã giúp hệ thống hoạt động ổn định và dễ dàng hơn trong việc bảo trì.

Tài liệu này sẽ cung cấp hướng dẫn chi tiết về:

* Cấu trúc file header (.hpp) và file source (.cpp).
* Quy tắc đặt tên class, method và property.
* Cách sử dụng namespace, static, và const.
* Cách viết comment theo chuẩn AUTOSAR Adaptive.
* Ví dụ cụ thể về tổ chức mã nguồn.

## **2. Cấu trúc file header (.hpp) và file source (.cpp)**

### **2.1 Cấu trúc file header (.hpp)**

File header trong C++ là nơi chứa các khai báo của class, method, và property để các file khác có thể sử dụng. File header đóng vai trò là "giao diện" giữa các module khác nhau trong chương trình và cần phải tuân thủ các quy tắc chặt chẽ về cấu trúc và đặt tên.

#### **Include Guards**

Mỗi file header đều phải có include guard để tránh bị include nhiều lần, gây ra lỗi biên dịch. Include guard thường được viết dưới dạng #ifndef, #define, và #endif theo tên file.

| #ifndef MOTOR\_CONTROL\_HPP #define MOTOR\_CONTROL\_HPP  // Nội dung khai báo của file header  #endif // MOTOR\_CONTROL\_HPP |
| --- |

#### **Phần mô tả (Documentation Block)**

Một phần quan trọng của tiêu chuẩn AUTOSAR Adaptive là việc thêm phần mô tả chi tiết vào đầu mỗi file, giúp người khác hiểu được chức năng và mục đích của file mà không cần đọc chi tiết mã. Các thông tin như tên file, phiên bản, ngày tạo và tên tác giả cần được ghi rõ.

| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \* @file MotorControl.hpp \* @brief Khai báo các hàm và class liên quan đến điều khiển động cơ. \* @version 1.0 \* @date 2024-09-12 \* @author HALA Academy \* @website https://hala.edu.vn \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
| --- |

#### **Khai báo class, method và property**

Trong file header, các class và method chỉ nên được khai báo, không định nghĩa. Các method trong class được khai báo ở đây để các file khác có thể sử dụng chúng.

| class MotorController { public:  void startMotor(int motorId); // Khai báo method khởi động động cơ  static void resetAllMotors(); // Khai báo method static để reset tất cả động cơ private:  int motorSpeed; // Property lưu tốc độ động cơ  static int totalMotors; // Property static lưu tổng số động cơ }; |
| --- |

### **2.2 Cấu trúc file source (.cpp)**

File source (.cpp) là nơi chứa phần định nghĩa của các method đã được khai báo trong file header. Để đảm bảo mã nguồn dễ đọc và bảo trì, file source cần tuân theo các quy tắc sau:

#### **Include file header**

File source cần include chính xác các file header tương ứng để sử dụng các khai báo class, method và property.

| #include "MotorControl.hpp" |
| --- |

#### **Phần mô tả (Documentation Block)**

Tương tự như file header, file source cũng cần có phần mô tả chi tiết về chức năng của file và thông tin tác giả, phiên bản.

| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \* @file MotorControl.cpp \* @brief Định nghĩa các method điều khiển động cơ. \* @version 1.0 \* @date 2024-09-12 \* @author HALA Academy \* @website https://hala.edu.vn \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
| --- |

#### **Định nghĩa method**

Trong file source, các method được định nghĩa chi tiết với phần thân thực hiện logic của chúng. Mỗi method cần có comment để giải thích chức năng, các tham số đầu vào, và giá trị trả về.

| void MotorController::startMotor(int motorId) {  motorSpeed = 100; // Đặt tốc độ cho động cơ }  void MotorController::resetAllMotors() {  totalMotors = 0; // Đặt lại tổng số động cơ  std::cout << "All motors have been reset." << std::endl; } |
| --- |

## **3. Quy tắc đặt tên class, method và property**

### **3.1 Quy tắc đặt tên class**

Tên class trong C++ theo tiêu chuẩn AUTOSAR Adaptive phải tuân theo quy tắc **PascalCase**, tức là mỗi từ đều bắt đầu bằng chữ in hoa. Tên class cần phải mô tả chính xác chức năng hoặc mục đích của nó.

| class MotorController { /\* ... \*/ }; |
| --- |

Ví dụ, class chịu trách nhiệm điều khiển động cơ nên được đặt tên là MotorController.

### **3.2 Quy tắc đặt tên method**

Tên method sử dụng quy tắc **camelCase**, nghĩa là chữ cái đầu tiên của tên method viết thường và mỗi từ tiếp theo bắt đầu bằng chữ in hoa. Tên method phải mô tả chính xác chức năng của nó để dễ hiểu và dễ bảo trì.

| void startMotor(int motorId); // Method khởi động động cơ void stopMotor(int motorId); // Method dừng động cơ |
| --- |

### **3.3 Quy tắc đặt tên property**

Tên property nên tuân theo quy tắc **camelCase**, bắt đầu bằng chữ thường và rõ ràng về chức năng của nó. Property static được chia sẻ giữa tất cả các đối tượng của lớp và thường được đặt rõ ràng trong file header.

| int motorSpeed; // Property lưu tốc độ động cơ static int totalMotors; // Property static lưu tổng số động cơ |
| --- |

## **4. Cách sử dụng namespace, static và const**

### **4.1 Sử dụng namespace**

Namespace giúp phân chia các phần khác nhau của chương trình thành các không gian tên riêng biệt, tránh xung đột tên giữa các module. Mỗi module hoặc nhóm chức năng nên có một namespace riêng.

| namespace Motor {  class MotorController {  // Định nghĩa các method và property liên quan đến motor  }; } |
| --- |

Trong ví dụ này, tất cả các khai báo liên quan đến Motor được tổ chức trong namespace Motor, giúp tránh xung đột với các module khác.

### **4.2 Sử dụng static**

Method static là phương thức của class có thể được gọi mà không cần tạo đối tượng. Method static chỉ có thể truy cập các property và method static khác trong cùng class. Nó thường được dùng để quản lý các tài nguyên hoặc dữ liệu chung cho tất cả các đối tượng của class.

| class MotorController { public:  static void resetAllMotors(); // Method static để đặt lại trạng thái tất cả động cơ private:  static int totalMotors; // Property static dùng chung giữa tất cả các đối tượng }; |
| --- |

### **4.3 Sử dụng const**

Sử dụng từ khóa const giúp đảm bảo rằng giá trị của property hoặc trạng thái của method không thể thay đổi sau khi được khởi tạo. Điều này giúp cải thiện độ an toàn của mã nguồn và ngăn ngừa các thay đổi không mong muốn.

| void startMotor(int motorId) const; // Method không thể thay đổi dữ liệu thành viên |
| --- |

const giúp đảm bảo rằng method này không thể thay đổi trạng thái của bất kỳ thành viên nào trong class, điều này rất hữu ích trong việc bảo vệ các thuộc tính quan trọng của đối tượng.

## **5. Cách viết comment chuẩn theo AUTOSAR Adaptive**

### **5.1 Comment trong file header**

Trong file header, comment giúp người đọc hiểu được chức năng của các class, method và property mà không cần phải đọc toàn bộ mã. Mỗi method cần được mô tả về chức năng, các tham số và giá trị trả về.

| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \* @file MotorControl.hpp \* @brief Khai báo các hàm và class liên quan đến điều khiển động cơ. \* @version 1.0 \* @date 2024-09-12 \* @author HALA Academy \* @website https://hala.edu.vn \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
| --- |

### **5.2 Comment trong file source**

Trong file source, comment phải giải thích rõ ràng các đoạn mã và các method, giúp người đọc hiểu được logic của chương trình. Mỗi method phức tạp cần có comment để giải thích mục đích và cách hoạt động của nó.

| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \* @brief Khởi động động cơ với một ID cụ thể. \* @param motorId ID của động cơ cần khởi động. \* @return void \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void MotorController::startMotor(int motorId) {  motorSpeed = 100; // Đặt tốc độ động cơ } |
| --- |

## **6. Ví dụ cụ thể về tổ chức mã nguồn**

#### **6.1 File Header: MotorControl.hpp**

| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \* @file MotorControl.hpp \* @brief Khai báo các method và class liên quan đến điều khiển động cơ. \* @details File này cung cấp giao diện điều khiển động cơ trong hệ thống nhúng. \* Class MotorController chứa các phương thức để khởi động, dừng động cơ, \* điều chỉnh tốc độ và lấy trạng thái động cơ. Các phương thức static được sử dụng \* để quản lý trạng thái chung của tất cả các động cơ. \* @version 1.0 \* @date 2024-09-12 \* @author HALA Academy \* @website https://hala.edu.vn \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #ifndef MOTOR\_CONTROL\_HPP #define MOTOR\_CONTROL\_HPP  #include <cstdint> #include <iostream>  // Định nghĩa giới hạn tốc độ tối thiểu và tối đa của động cơ #define MAX\_MOTOR\_SPEED 100 #define MIN\_MOTOR\_SPEED 0  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \* @class MotorController \* @brief Class đại diện cho bộ điều khiển động cơ. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ class MotorController { public:  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* @brief Khởi động động cơ với ID cụ thể.  \* @param motorId ID của động cơ cần khởi động.  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void startMotor(int motorId);   /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* @brief Dừng động cơ với ID cụ thể.  \* @param motorId ID của động cơ cần dừng.  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void stopMotor(int motorId);   /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* @brief Đặt tốc độ cho động cơ.  \* @param motorId ID của động cơ cần đặt tốc độ.  \* @param speed Tốc độ cần đặt, trong khoảng MIN\_MOTOR\_SPEED - MAX\_MOTOR\_SPEED.  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void setMotorSpeed(int motorId, int speed);   /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* @brief Lấy tốc độ hiện tại của động cơ.  \* @param motorId ID của động cơ cần kiểm tra.  \* @return int Trả về tốc độ hiện tại của động cơ.  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  int getMotorSpeed(int motorId) const;   /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* @brief Lấy trạng thái của động cơ (ON/OFF).  \* @param motorId ID của động cơ cần kiểm tra.  \* @return bool Trả về true nếu động cơ đang chạy, false nếu đã dừng.  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  bool isMotorRunning(int motorId) const;   /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* @brief Đặt lại trạng thái của tất cả các động cơ.  \* @details Đặt lại tất cả động cơ về trạng thái ban đầu (dừng và tốc độ = 0).  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  static void resetAllMotors(); // Phương thức static quản lý tất cả động cơ   /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* @brief Lấy tổng số động cơ hiện có trong hệ thống.  \* @return int Trả về tổng số động cơ.  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  static int getTotalMotors(); // Phương thức static trả về tổng số động cơ hiện có  private:  int motorSpeed; // Tốc độ hiện tại của động cơ  bool motorState; // Trạng thái động cơ: true = ON, false = OFF  static int totalMotors; // Tổng số động cơ trong hệ thống (biến static) };  #endif // MOTOR\_CONTROL\_HPP |
| --- |

#### **6.2 File Source: MotorControl.cpp**

| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \* @file MotorControl.cpp \* @brief Định nghĩa các method điều khiển động cơ. \* @details File này chứa định nghĩa các method được khai báo trong \* MotorControl.hpp, bao gồm khởi động, dừng động cơ, đặt tốc độ, \* và kiểm tra trạng thái của động cơ. Ngoài ra, còn có các phương thức \* static để quản lý tổng số động cơ trong hệ thống. \* @version 1.0 \* @date 2024-09-12 \* @author HALA Academy \* @website https://hala.edu.vn \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #include "MotorControl.hpp"  // Khởi tạo biến static int MotorController::totalMotors = 0;  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \* @brief Khởi động động cơ với ID cụ thể. \* @details Phương thức này sẽ thay đổi trạng thái của động cơ sang ON và \* thiết lập tốc độ mặc định cho động cơ. \* @param motorId ID của động cơ cần khởi động. \* @return void \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void MotorController::startMotor(int motorId) {  motorSpeed = 50; // Đặt tốc độ mặc định  motorState = true; // Đặt trạng thái là ON  totalMotors++; // Tăng số động cơ đang chạy  std::cout << "Motor " << motorId << " started at speed " << motorSpeed << "." << std::endl; }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \* @brief Dừng động cơ với ID cụ thể. \* @details Phương thức này sẽ thay đổi trạng thái của động cơ sang OFF. \* @param motorId ID của động cơ cần dừng. \* @return void \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void MotorController::stopMotor(int motorId) {  motorSpeed = 0; // Đặt tốc độ về 0  motorState = false; // Đặt trạng thái là OFF  totalMotors--; // Giảm số động cơ đang chạy  std::cout << "Motor " << motorId << " stopped." << std::endl; }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \* @brief Đặt tốc độ cho động cơ. \* @details Phương thức này kiểm tra xem tốc độ truyền vào có hợp lệ hay không \* (trong khoảng từ MIN\_MOTOR\_SPEED đến MAX\_MOTOR\_SPEED). \* Nếu hợp lệ, tốc độ của động cơ sẽ được cập nhật. \* @param motorId ID của động cơ cần đặt tốc độ. \* @param speed Tốc độ cần đặt. \* @return void \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void MotorController::setMotorSpeed(int motorId, int speed) {  if (speed < MIN\_MOTOR\_SPEED || speed > MAX\_MOTOR\_SPEED) {  std::cout << "Error: Speed out of range for motor " << motorId << "." << std::endl;  return;  }  motorSpeed = speed; // Cập nhật tốc độ động cơ  std::cout << "Motor " << motorId << " speed set to " << motorSpeed << "." << std::endl; }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \* @brief Lấy tốc độ hiện tại của động cơ. \* @details Trả về tốc độ hiện tại của động cơ có ID cụ thể. \* @param motorId ID của động cơ cần kiểm tra. \* @return int Trả về tốc độ hiện tại của động cơ. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ int MotorController::getMotorSpeed(int motorId) const {  std::cout << "Motor " << motorId << " current speed: " << motorSpeed << "." << std::endl;  return motorSpeed; }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \* @brief Lấy trạng thái của động cơ (ON/OFF). \* @details Kiểm tra trạng thái hiện tại của động cơ có ID cụ thể. \* @param motorId ID của động cơ cần kiểm tra. \* @return bool Trả về true nếu động cơ đang chạy, false nếu đã dừng. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ bool MotorController::isMotorRunning(int motorId) const {  std::string status = motorState ? "running" : "stopped";  std::cout << "Motor " << motorId << " is currently " << status << "." << std::endl;  return motorState; }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \* @brief Đặt lại trạng thái của tất cả các động cơ. \* @details Đặt lại tất cả động cơ về trạng thái ban đầu (dừng và tốc độ = 0). \* Phương thức này sẽ thiết lập lại tất cả các biến động cơ. \* @return void \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void MotorController::resetAllMotors() {  totalMotors = 0; // Đặt lại tổng số động cơ  std::cout << "All motors have been reset." << std::endl; }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \* @brief Lấy tổng số động cơ hiện có trong hệ thống. \* @details Phương thức này trả về tổng số động cơ hiện đang hoạt động trong hệ thống. \* @return int Trả về tổng số động cơ hiện có. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ int MotorController::getTotalMotors() {  std::cout << "Total number of motors running: " << totalMotors << "." << std::endl;  return totalMotors; } |
| --- |