**Lab Exercise 8 – Singleton Design Pattern in C++**

Let's create a simple example to illustrate the difference between using and not using the Singleton design pattern in an embedded system.

**Without Singleton:**

#include <iostream>

// Class representing a Configuration Manager

class ConfigurationManager {

public:

// Dummy configuration data

void configureSystem() const {

std::cout << "Configuring embedded system..." << std::endl;

// Simulate configuration logic

}

};

int main() {

// Without Singleton - Creating multiple instances of ConfigurationManager

ConfigurationManager configManager1;

configManager1.configureSystem(); // Configure the embedded system

ConfigurationManager configManager2; // Another instance

configManager2.configureSystem(); // Configure the embedded system again

return 0;

}

**With Singleton:**

#include <iostream>

// Singleton class - ConfigurationManager

class ConfigurationManager {

private:

// Private constructor to prevent instantiation

ConfigurationManager() {}

// Instance of the singleton

static ConfigurationManager\* instance;

public:

// Public method to get the singleton instance

static ConfigurationManager\* getInstance() {

if (!instance) {

instance = new ConfigurationManager();

}

return instance;

}

// Dummy configuration data

void configureSystem() const {

std::cout << "Configuring embedded system..." << std::endl;

// Simulate configuration logic

}

};

// Initialize the static instance variable

ConfigurationManager\* ConfigurationManager::instance = nullptr;

int main() {

// With Singleton - Accessing the ConfigurationManager singleton instance

ConfigurationManager\* configManager1 = ConfigurationManager::getInstance();

configManager1->configureSystem(); // Configure the embedded system

ConfigurationManager\* configManager2 = ConfigurationManager::getInstance();

configManager2->configureSystem(); // Configure the embedded system again

// Check if both instances are the same

if (configManager1 == configManager2) {

std::cout << "Both instances are the same. Singleton pattern is working." << std::endl;

} else {

std::cout << "Singleton pattern is not working correctly." << std::endl;

}

return 0;

}

**Observations:**

In the version without Singleton, multiple instances of ConfigurationManager are created, which may lead to redundant configurations and potential issues in managing global resources in an embedded system.

In the version with Singleton, the ConfigurationManager class is designed as a Singleton, ensuring that there is only one instance globally accessible through the getInstance method. This prevents redundant configurations and provides a single point of control over the configuration settings.

In an embedded system, using the Singleton pattern can help ensure that certain resources or configurations are managed consistently across the entire system