Shared Data Initialization Solutions

Shared Data

- Give a list of all the ways to create shared data in C++
  - Global variable
  - Static variable at namespace scope
  - Class member which is declared static
  - Local variable which is declared static

Shared Data Initialization

- How are these shared data variables initialized?
  - Global variable
    - Initialized at program startup
  - Static variable at namespace scope
    - Initialized at program startup
  - Class member which is declared static
    - Initialized at program startup
  - Local variable which is declared static
    - Initialized when the program reaches its declaration

Shared Data Initialization

- Does these initialization procedures have any implications for thread safety?
  - Global variable
    - Initialized when only one thread is running - thread-safe
  - Static variable at namespace scope
    - Initialized when only one thread is running - thread-safe
  - Class member which is declared static
    - Initialized when only one thread is running - thread-safe
  - Local variable which is declared static
    - Initialized when many threads could be running - data race?

Static Local Variable Initialization Before C++11

- Before C++11, could static local variables be initialized in a thread-safe way?
  - No support from the language
  - May have been possible using a (third-party) mutex
  - Inefficient

Static Local Variable Initialization in C++11

- Has the initialization of static local variables changed in C++11?
  - Yes
  - The first thread to reach the declaration initializes the variable
  - No other thread is allowed to initialize the variable
    - Not even if it reaches the declaration concurrently with the first thread
  - The implementation synchronizes the threads which initialize the variable
  - There is no data race
- What happens if the variable is modified after initialization?
  - The same rules apply as to non-static variables
  - The accesses need to be synchronized to avoid a data race

Classic Singleton Implementation

- What is meant by a Singleton class?
  - A Singleton is a class which has only one global instance in the program
- Briefly describe the traditional implementation of a Singleton class
  - The constructor is declared private
  - The copy and move operators are deleted
  - A static member function returns the instance of the class
  - If the instance has not been initialized, the function initializes it before returning it
  - Otherwise, the function returns the instance directly

Classic Singleton Implementation

- Implement a traditional Singleton class
- Write a multi-threaded program to test it
- Is the traditional Singleton thread-safe?
  - No
  - Threads can concurrently access the static pointer member
  - These threads can modify the static pointer member
  - Unless these accesses are synchronized, there is a data race

C++11 Singleton Implementation

- Does C++11 give us a better way to implement the Singleton?
  - Yes
  - We use a static local variable to store the unique instance
  - In C++11, this will be initialized in a thread-safe way
- Implement a Singleton class which uses these features
- Write a multi-threaded program to test your class