Data Structures and Concurrency Solutions

Modifying Operations

- A data structure is accessed by two threads concurrently
- For each of these scenarios, state whether there is a potential data race
- Explain your answers
  - Thread A writes to the first element of a linked list and thread B reads the second element
  - No data race. The data in the two nodes has different addresses in memory
  - Thread A removes the first element of a single-linked list and thread B reads the second element
  - No data race. Removing a node does not affect the data in a different node
  - Thread A removes the second element of a double-linked list and thread B reads the first element
  - No data race. Removing a node does not affect the data in a different node

Modifying Operations

- Thread A removes the first element of a single-linked list and thread B removes the second element
- Data race. Thread B needs to update the first node's "next" pointer. This could happen after thread A has deleted the first node.
- Thread A removes the second element of a double-linked list and thread B removes the first element
- Data race. Thread A needs to update the first node's "next" pointer. This could happen after thread B has deleted the first node. Similarly, thread B could update the second node's "previous" pointer after thread A has deleted it
- Thread A removes the second element of a double-linked list and thread B removes the fifth element
- No data race. Removing a node only affects its immediate neighbours

Basic Thread Safety Guarantee

- When two threads can access the same STL container object, what guarantees do we have?
  - It is safe for the threads to concurrently access the object, provided they do not modify it
  - It is safe for one thread to modify the object, provided no other threads access the object concurrently
  - Depending on the type of the container, it may be safe to make concurrent modifications to different parts of the object
- Write a program which demonstrates unsafe access to an STL container object by threads
- Alter your program so that it is now thread-safe

Coarse-grained Locking

- What is meant by "coarse-grained" locking?
  - Any thread that accesses an element of the object must lock all the elements
- What are the advantages of coarse-grained locking?
  - Does not require any special code in the object
  - Can be used with types which were designed for single-threaded programs
- What are the disadvantages of coarse-grained locking?
  - Prevents concurrent access to the object, even when it would be safe to do so

Fine-grained Locking

- What is meant by "fine-grained" locking?
    - When a thread that accesses an element of the object, only the elements which will be affected by it are locked
- What are the advantages of fine-grained locking?
    - Allows concurrent access to the object, except when it would be unsafe
- What are the disadvantages of fine-grained locking?
    - Requires extra code in the object
    - Difficult to get right