Unique Pointers and Custom Deleters Solutions

unique_ptr and pointer management

- Why is unique_ptr useful for managing pointers which were not returned by new()?
  - Uses RAII idiom with ownership semantics
  - Adopts pointer in constructor
  - Performs clean-up in destructor
  - Can be moved but not copied, correctly handles transfer of ownership of pointer
- Why can this be problematical?
  - By default, unique_ptr's destructor calls delete()
  - If the pointer was not returned by calling new(), there will be a memory error
- Write a program which demonstrates this problem

Custom deleters

- What is a deleter?
  - A deleter is a function which is called by the unique_ptr destructor instead of delete()
  - It is used to release the resource which was acquired in the constructor

unique_ptr with deleter

- How do we use a deleter with unique_ptr?
  - We give the type of the deleter as an optional second template parameter
  - We pass the deleter as an optional second argument to the constructor of unique_ptr

Deleter Type

- What happens if we do not know the type of the deleter?
  - If we do not know the type of the deleter, we can use decltype(deleter_name)
  - The compiler will replace this by the type of the deleter

Custom deleter

- Write a program which uses a custom deleter
- What would you expect to happen if an exception is thrown?
  - If an exception is thrown, the unique_ptr destructor is called
  - The destructor calls the deleter and releases the resource
- Check that your program behaves as expected when an exception is thrown