CSCI251/CSCI851 Autumn-2020
Advanced Programming (**EMC**)

Effective Modern C++
Scott Meyers

“A list …

- … of 42 specific ways to improve your use of C++11 and C++14.”
- These slides are effectively just the table of contents for the book.
- Some of this is beyond the scope of this subject.

Part 1: Deducing types

1. Understand template type deduction.
2. Understand `auto` type deduction.
3. Understand `decltype`.
4. Know how to view deduced types.

Part 2: `auto`

5. Prefer `auto` to explicit type declarations.
6. Use the explicitly typed initializer idiom when `auto` deduces undesired types.

Part 3: Moving to Modern C++

7. Distinguish between () and {} when creating objects.
8. Prefer `nullptr` to `0` and `NULL`.
9. Prefer alias declarations to `typedef`s.
10. Prefer scoped `enum`s to unscoped `enum`s.
11. Prefer deleted functions to private undefined ones.
12. Declare overriding functions `override`.

13. Prefer `const_iterators` to `iterators`.
14. Declare functions `noexcept` if they won't emit exceptions.
15. Use `constexpr` whenever possible.
16. Make `const` member functions thread safe.
17. Understand special member function generation.

Part 4: Smart Pointers

18. Use `std::unique_ptr` for exclusive-ownership resource management.
19. Use `std::shared_ptr` for exclusive-ownership resource management.
20. Use `std::weak_ptr` for `std::shared_ptr`-like pointers that can dangle.
21. Prefer `std::make_unique` and `std::make_shared` to direct use of `new`.
22. When using the Pimple Idiom, define special member functions in the implementation file.

Part 5: Rvalue references, Move Semantics, and Perfect Forwarding

23. Understand `std::move` and `std:forward`.
24. Distinguish universal references (diff name used now) from rvalue references.
25. Use `std::move` on rvalue references, `std::forward` on universal references.
26. Avoid overloading on universal references.
27. Familiarise yourself with alternatives to overloading on universal references.

28. Understand reference collapsing.
29. Assume that move operations are not present, not cheap, and not used.
30. Familiarise yourself with perfect forwarding failure cases.

Part 6: Lambda functions

31. Avoid default capture modes.
32. Use init capture to move objects into closures.
33. Use `decltype` on `auto&&` parameters to `std::forward` them.
34. Prefer lambdas to `std::bind`.

Part 7: The Concurrency API

35. Prefer task-based programming to thread-based.
36. Specify `std::launch::async` if asynchronicity is essential.
37. Make `std::threads` unjoinable on all paths.
38. Be aware of variable thread handle destructor behaviour.
39. Consider void futures for one-shot event communication.

Part 8: Tweaks

41. Consider pass by value for copyable parameters that are cheap to move and always copied.
42. Consider emplacement instead of insertion.