Workshop Guidelines

1. Style - header file comments should be present in `*.h`
2. Style - header file comments should be present in `*.cpp`

   Example

```
// Workshop ? – short title
// event.h – file name
// Chris Szalwinski – author
// 123456789 – author id
// other specified information
```

3. Style - remove TODO comments after instructions followed
4. File Preliminaries - `#pragma once` or `#ifndef` but not both
5. File Preliminaries - use `< >` for system files instead of `" " – < >` searches system directories first, then current directory
6. File Preliminaries - use `<cstring>` instead of `<string.h> –` in `std` namespace rather than global namespace
7. C++ Standard – prefer `#ifndef` over non-standard `#pragma once`
8. C++ Standard - do not start macros with _
9. C++ Standard - C++ disallows anonymous struct – you can't define a constructor
10. Type Safety – Initialization - prefer `{2}` to `= 2 after C++11`

```
int x {4.1}; // error: narrowing
int x = 4.1; // ok: x becomes 7 (truncation)
```

`{}` can be used for nearly all forms of initialization – often called uniform initialization – accepts explicit constructors – gives direct initialization

`={}` gives copy initialization – does not accept explicit constructors

```
struct A {explicit A(){}};
A a{}; // ok: direct initialization
A a = {}; // error: copy initialization
```

11. Type Safety – appropriate use of const: `X& src -> const X & src`

```
set(char -> set(const char
```

12. Type Safety – be consistent with type unsigned = unsigned value
13. Type Safety – `display()` should be `const`
14. Type Safety – Prefer const size_t M… = ?? to #define M… ?? for magic numbers
15. Type Safety – prefer Initialization, better than in ctor body also more efficient
16. Type Safety – use ctor instead of const_cast
17. Type Safety – Use const wherever possible

18. Type Safety – Use const and & wherever possible
19. Encapsulation - Data members should be private
20. Encapsulation – Private functions private should be private
21. Encapsulation – Class variables should be private as per specs
22. Encapsulation – If you have a public display your << operator doesn't need to be a friend
23. Readability – #include only those header files needed to identify names in .h file
24. Readability – #include only those header files needed to complete logic in .cpp file
25. Readability – use `setfill('0')` instead of hard coding leading 0s
26. Readability – `void display() const` is more readable than `auto display()->void`
27. Readability – prefer same type
28. Readability – Use a default parameter rather than a separate function definition
29. Readability – Avoid duplication in copy ctor and assignment operator
30. Readability - Avoid magic numbers – use const size_t MAGIC_NUMBER = 10;
31. Readability – Macros are typically all caps
32. Encapsulation – enforce strong encapsulation - avoid exposing more than necessary
33. Dynamic Memory Management – need copy constructor, assignment operator, destructor
34. Dynamic Memory Management – if no default initialization add `xxx = nullptr;` before calling assignment operator in copy constructor
35. Dynamic Memory Management – can call `delete [] x` if `x` is `nullptr` – no effect
36. Dynamic Memory Management – No need for `= nullptr` if default initialized
37. Dynamic Memory Management – missing deallocation in assignment operator
38. Dynamic Memory Management – missing deallocation in
39. Dynamic Memory Management – Check for self-assignment in assignment operator
40. Dynamic Memory Management – Copy ctor bug - need to initialize resource address to nullptr
41. Dynamic Memory Management – Move ctor bug - need to initialize resource address to nullptr
42. Dynamic Memory Management – Move operator should copy address not the data
43. Dynamic Memory Management – Memory leak in move assignment – deallocation missing
44. Dynamic Memory Management – If same object move or copy do not deallocate
45. Dynamic Memory Management – Need to deallocate memory in copy assignment
46. Dynamic Memory Management – Need to deallocate memory in move assignment
47. Dynamic Memory Management – Destructor missing
48. Scope - use `static` (internal linkage) for counter

49. Good practice - Do not use using namespace in .h files
50. Good practice - Add noexcept only if function will not throw an exception – not the case for memory allocation functions
51. Good practice - Avoid code duplication in assignment and constructor
52. Good practice - Don't need to close file – only need to clear its state and rewind
53. Good practice - Add `noexcept` to move constructor and assignment operator
54. Good Practice - No need for << to be a friend since it calls display which is a public member function - << should be a free helper as per specs
55. Good practice - use ctor instead of adding set function when specs limit the interface
56. Good practice - Shorten code by default initialization of pointers and instance variables
57. Good practice - use constexpr instead of const for macros
58. Good practice - avoid magic numbers – define a `constexpr unsigned M_WORDS{6}`
59. Lifetime - Object passed to move constructor must be near the end of its lifetime, it does not need to be a temporary object
60. Beyond Scope - `main` prototype missing in w1_p2
61. Instructions - Public member function added not in specs – could be protected