PRACTICE

HW1: Items 1-3

HW2: Items 1-7

HW3: Items 1-8

HW4: Items 1-11

HW5: Items 1-11

HW6: Items 1-18

HW7: Items 1-26

HW8: Items 1-26

HW9: Items 1-26

HW10: Items 1-28

1. Do not #include .cpp files
2. Include return 0 at the end of main
3. Use '\n' instead of std::endl whenever possible
4. Initialize a variable where it is defined or on the next line (unless it has a default constructor)
5. Initialize a variable close to where it is first used
6. Avoid C-style casts: be explicit about type of cast
7. Avoid global variables
8. Use size_t for all sizes and container indices and whenever function inputs/outputs are of that type
9. Put braces around all control flow bodies, even if just a single statement
10. Avoid bloating within reason (don't have an excessive number of if/elses or manually construct several values if the same logic could be done with one simple loop, etc.)
11. General const correctness (keeping const variables const, keeping loop references const where appropriate, etc.)
12. Const correctness for function arguments (marking inputs as const where appropriate)
13. Efficiency for function arguments (accepting references when appropriate)
14. General efficiency in variables (using references in loops where appropriate, etc.)
15. Use prefix ++ for efficiency in loops with iterators: for (auto itr = start; itr != past_end; ++itr) – where itr is an iterator
16. No bare using declarations or directives in header files
17. Use nullptr not NULL or 0 for a null pointer
18. Ensure std::rand is only seeded once if at all

19. Initialize all member variables of a class (unless default initialization is valid)
20. Use constructor initializer lists as much as possible in class construction
21. In the constructor initializer list, initialize members of a class in the order they are listed within the interface
22. Const correctness for member functions (accessors)
23. Proper encapsulation: use public/private appropriately and don't provide unnecessary getters/setters or access to class members
24. General design (ensure classes are usable – do not write a default constructor if it doesn't make sense, do not write getters/setters for all member variables unless required, etc.)
25. Efficiency for function return values (returning references when appropriate)
26. Use #ifndef header guards for header files, not #pragma once
27. Check a file stream is in a valid state before use
28. Close a file stream after use (unless its enclosing scope ends immediately after its use)

READABILITY

HW1: Items 1-2

HW2: Items 1-4

HW3: Items 1-4

HW4: Items 1-7

HW5: Items 1-7

HW6: Items 1-10

HW7: Items 1-10

HW8: Items 1-12

HW9: Items 1-12

HW10: Items 1-12

1. Detailed comments: explaining what/why, idea by idea
2. General layout (not cramped together, not excessive space, using new lines to reduce length of a line of code, putting blank spaces between for loop statements, etc.)
3. No magic numbers
4. Descriptive variable names

5. Commenting each branch of control flow, each if, else if, else, for, while, do, switch, …
6. Avoid redundancies in bools, e.g., if(full) is fine, if (full == true) is not okay
7. For empty statements in control flow, clearly document the intent, e.g. while(f(x++) < 8) {/* empty */ }
8. Function documentation must be of the precise Doxyen form:
   /**
   Description of function
   …
   @param x description of first parameter (if any)
   …
   @return description of what is returned (if a return)
   */
   and it should be done at the function declaration.
   Note that for templates, there should be an additional
   @tparam T description of T
   etc. for each template parameter.
9. Declare functions/constructors in .h files and define them in .cpp files (unless templates are involved) – the /** */ documentation must be in the header files not cpp files
10. Do not abuse auto making code harder to read
11. All classes should be document with a Doxyen tag:
    /**
    @class class_name description of class
    */
12. Use “class” for classes where encapsulation is important; use “struct” for simple classes.