1. Meaningful Names

Use function, variable and class names that describe their operation by name, making it easier to navigate through the code. It is important to avoid one-letter names or ambiguous abbreviations

2. Comments

Try to write your code to avoid the need to add comments, the code should be largely self-describing. You can use comments to separate specific areas to make it easier to find. If there is a need to use comments, use them to explain non-obvious or complex decisions or logic.

3. Functions

Remember to make functions/methods small and focused on a specific responsibility (Single Responsibility Principle). It is important to aim for a function to have no more than 20 lines of code, of course this is not always possible but it is important to keep this in the back of your mind. Use a nomenclature that explains the function itself.

4. Formatting

Use consistent formatting and adhere to the chosen style. Try to keep any tabs or spaces the same across all files. At the same time, try to keep the code readable

5. Error Handling

Use exceptions only in exceptional situations, do not use them to call up information in the console that a function has worked or is working. When using exceptions, provide clear information about what went wrong

6. STL Usage

Be familiar with and use the STL C++ Standard Library. Don't forget container classes for example vector. Try to use the functions available in them to improve the writing of algorithms, functions or methods

7. Class Design
   Follow the principle of one responsibility per class and prefer composition over inheritance. Use access specifiers such as public, private and protected to avoid unwanted use of or access to a variable/method/function
8. Refactoring

When writing code, even when we try to keep it clean we sometimes make a mess of it making the code less readable. Remember to refactor your code to improve code readability.

9. Testing

For critical functionality, write unit tests to eliminate potential bugs in the code. Unit tests will support your work on the code and, before the code goes into production, help you to improve the performance of the program

10. Continuous Integration

Use const to prevent messy overwriting of data in a variable as well as to ensure its immutability. Correct use of const will help you avoid getting messy erroneous data or results.