Safety and coding habits

- We want to develop sound coding habits to minimize possible future issues
- Bugs happen anyway: know it, be prepared for it, write code to minimize the run-time impact
- Many of the ideas discussed here may actually be part of the code standards for a project/team/organization

Simplify, decompose, abstract

- Keep computations and logic statements simple and clear
- If you have a complex computation, break it up into smaller, clearer steps (that are easier to understand and much less likely to contain subtle bugs)
- Use clear comments and clear, informative identifiers
- When you change the code, be sure you keep comments up to date

Compilation

- Always use your compiler’s error checking options, e.g. for g++ use -Wall -Wextra (all warnings + extra warnings)
- All warnings reveal code weaknesses: when your compiler generates warnings, ***fix them***
- If you get in the habit of ignoring streams of warnings because “those ones aren’t important”, then you’re going to miss buried warnings that are important (and other devs are going to think of your code as sloppy/error-prone)

Return values

- If there is a chance that a function will be unable to work correctly (e.g. if passed invalid data), include a return value (or reference parameter) to reflect the completion status
- Functions use return values for a reason, don’t ignore them
- When you call a function that returns a value, check that value to make sure it reflects that things worked correctly
- This also applies to pass-by-reference parameters – ensure that whatever was done to a ref parameter reflects a valid value/change

Prevent input errors

- When getting data from a user, try to maximize the chance the data will be entered correctly
- Give the user a clear prompt that reflects what they're supposed to be entering, the input format, the units being used (miles vs kilometres vs furlongs etc), etc
- Pick an input mechanism that prevents errors (e.g. get a date through a calendar pop-up, not as plain text)

Check for errors asap

- Ideally, check input for errors as close to the time/point of entry as possible, giving the opportunity to fix the error or alter processing accordingly
- Minimize the impact of detected errors: e.g. if one field of a form is incorrect then don’t make them re-enter the entire form
- Remember that input from files or other programs also needs to be checked for potential errors

Anticipate common errors

- Check parameters for common issues (null pointers, out of range values, etc)
- Initialize variables when you declare them
- Consider bounds-checking for array accesses
- Consider checking for runaway recursion or infinite loops

Code standards

- We’ll discuss standards in detail later, but choosing and following reasonable standards can make it much easier to read, understand, and maintain your code and other developers’ code