Constexpr If Solutions

Constexpr If

- Briefly describe constexpr if
  - constexpr if evaluates conditionals at compile time
  - It also conditionally compiles code - only the code in the "true" branch is compiled into the program

Runtime If Statement

- Explain why the following code does not compile

```
template<typename T>
string get_string(const T& arg) {
   if (std::is_same<std::string, T>::value)      // Replaced by "true" if T is a string, else "false"
      return arg;
   else
      return to_string(arg);
}
```

Runtime If Statement

- Explain why the following code does not compile
  - This code uses an if statement which will be evaluated at runtime
  - When the compiler checks the instantiated template, it must assume that either branch can be taken
  - Both branches must compile
  - If T is not a string, the first branch does not compile
  - If T is a string, the second branch does not compile

Constexpr If Statement

- Explain how constexpr if can solve this problem
  - With constexpr if, the statement will be evaluated at compile time
  - Only one of the branches will be present in the program's source code after the template is instantiated
  - When the compiler checks the instantiated template, it will only see the branch which is taken, which will always compile
- Write a simple program to check your solution

Advantages of Constexpr If

- List some alternatives to using constexpr if and explain the advantages of using constexpr if
  - Preprocessor directives such as #if perform textual substitution
    - No understanding of C++ syntax or types
  - Template Specialization
    - Requires multiple functions which need to be in a specific order
    - Not suitable for complex tests
  - SFINAE and enable_if
    - Complex and obscure code
    - Incomprehensible error messages
  - constexpr if
    - Single function with normal-looking code
    - Normal-looking error messages