Templates Overview Solutions

Writing a template

- Write a template function which takes two arguments of a generic type and returns the greater of the two arguments
- Write a program which calls your function and passes arguments of type double
- Write out the code for the function which is instantiated by the compiler

Templates and Code Organization

- The normal practice is to put function declarations into a header file, which is included by any source code files which call those functions. The full function definition is put in a separate source code file
- Does this method of code organization work with template functions?
  - The full function definition must be available to the compiler when the function is called
  - The compiler needs to have access to the function body to be able to instantiate the template
  - Usually the full template function definition is written in the header file
  - Sometimes the template function definitions are put in a separate file, which is included after the header file

Class Template

- Write down the definition of a template class whose members are
  - A member called "data", which has the same type as the template parameter
  - A constructor which initializes "data" from an argument of the same type

```
template <class T>
class Test {
   T data;
   Test(const T& data) : data(data) {}
};
```

Class Template

- Write down code to create an instance of this class with std::string as the parameter and the string "Hello" as the initial value of the member

```
Test<string> test{"Hello"};
```

- Write out the class definition generated by the compiler for this instantiation

```
class Test_xcajkjha {                                  // Instantiated  with unique name
   string data;
   Test(const string& data) : data(in_data) {}
};
```

Constructor Argument Deduction in C++17

- Without using explicit template parameters, write a simple program that creates an std::vector<int>
- Check that your code compiles