Modern C++ Overview
Part Four Solutions

Rvalue references and overloading

- Write a function which takes an int by const reference and an overloaded version which takes an int by rvalue reference
  - Each function displays its argument type
- Write a program which calls the overloaded function and passes
  - An int variable
  - The result of calling std::move() on an int variable
  - An integer literal

Rvalue references and overloading

- Explain your results
  - An int variable is an lvalue (we can take its address), so the lvalue overload is called
  - std::move() casts its argument to an rvalue, so the rvalue overload is called
  - A literal is an rvalue (we cannot take its address), so the rvalue overload is called

Move-only types

- What is meant by a move-only type? Give an example of a move-only type
  - A move-only type cannot be copied, but it can be moved
  - Examples: fstream, unique_ptr
- Why are move-only types useful?
  - Move-only types are useful when a class owns a resource and manages its lifetime. In order to give an object sole ownership of the resource, copying cannot be allowed. However, the resource can be transferred from one object to another, in which case the target object becomes the new owner of the resource.

Pass by move

- What property must a class have in order that objects of that class can be passed by move?
  - The class must define a move constructor

Move operators

- Write down the prototypes of the move constructor and move assignment operator of a class called "Test"

```
Test(Test && other) noexcept;                    // Move constructor
Test& operator=(Test && other) noexcept;         // Move assignment operator
```

Deleted and Defaulted Operators

- Describe what effect the "delete" and "default" keywords have in the following statements

```
Test(const Test& other) = delete;
Test(const Test& other) = default;
```

  - "delete" means that the copy constructor for this class cannot be called. This prevents objects of this class from being copied
  - "default" means that the compiler will generate a default copy constructor which calls the copy constructor of all the data members of the class

Deleted and Defaulted Operators

- Why are these keywords useful?
  - "delete" allows us to write a class that cannot be copied (but could still be moved)
  - "default" saves us from having to write our own default special member function, if that is all we want. (Sometimes this is necessary; for example, if a copy constructor is defined, the compiler will not generate a move constructor). This avoids errors and the need to maintain a hand-written function
  - "default" also helps document the code, even if the compiler would have generated it anyway. It is not always immediately obvious which special member functions will be generated, particularly in derived classes

Class which can be Moved but not Copied

- Write a class which can be moved but not copied, using the "delete" and "default" keywords where appropriate
- Write a program which creates objects of this class. Demonstrate that move operations are allowed, but copying objects is not

Pass by value... or by move?

- Write a function which takes a vector of std::string by value and prints out the number of elements in the vector
- Write a program which creates a vector of std::string and passes it to the function
- Display the number of elements in the vector
  - Before making the call
  - After returning from the call
- Modify your program so that the vector is passed as an rvalue

Pass by value... or by move?

- Explain your results
  - In the first case, the vector is an lvalue and is passed by value. The argument to the function will be a copy of the vector in main(). All three print statements will give the same number of elements
  - In the second case, the vector is an rvalue and is passed by move. The data in the vector in main() will be moved into the argument to the function, leaving an empty object. The final print statement will give the number of elements as zero