Part A
Most of the multiple choice was done fairly well. The questions with the worst performance in terms of percentage getting the correct answer were: 2, 3, 4, 7, 14 and 21.

Part B

Q1
All or nothing since I'd said in lecture it'd be on the exam. Many students couldn't spell "Acquisition" correctly. We accepted "Initialisation" as an alternate to "Initialization".

Q2
All or nothing since I'd said in lecture it'd be on the exam. We accepted "Modelling" as an alternate to "Modeling".

Q3
Public accessors and mutators wouldn't be correct since they can't be called if they're not public. Just mentioning exposing information in the header file isn't a complete answer since the language forces you to do that unless you want to implement Pimpl everywhere. We talked about three different forms of representation exposure in lecture.

Q4
Some students just said the destructor should be implicitly invoked when a stack-allocated object goes out of scope, and the program would terminate immediately, but didn’t mention stack unwinding, two exceptions at the same time, memory leaks, etc.

Q5
The most common mistake is that students didn’t strike through the declaration of vector words, and some students put the implementation class definition in the Dictionary class rather than a forward declaration, which would expose the implementation and thus not correctly solve the problem.

Q6
Many students differentiated Adapter and Facade by the number of classes involved. Please review pages Chapter 7, pages 258-260 of "Head First Design Patterns" in the assigned readings, the section online listed as "Lights, Camera, Facade!".

Q7
Some students said the adapter pattern was used in MVC pattern. Or just mentioned inheritance, which isn't enough since all design patterns use inheritance as part of programming to the interface rather than the implementation.

Q8
Many students gave reasons for using copy-swap that were true, but weren't the main benefit (e.g. less code, maintainability, etc.). We were explicitly looking for the idea of exception safety i.e. object left in original state if the deep copy failed.

Part C

Q3
Most students did fairly well on parts a) and b). For part a), we accepted any four reasonable representation invariants (see the solution set for the list). For part b), the commonest mistakes were to:

- not require user to be a valid address i.e. not nullptr, often leaving the requires clause blank,
- specify that root_ is modified, which is true only if the graph is initially empty or the new user’s email address come alphabetically first,
• really should specify what causes the exception to be thrown, even though the exception name was pretty meaningful, and
• forgetting to specify in the ensures clause that the elements of the graph that were previously there are still there, and that the list's order is maintained i.e. the user is inserted into the correct location.

Very few did well on part c) since almost nobody gave an actual abstraction function. Either the answer was blank, or they just provided the abstraction and the representation but not the actual description/function of how to go from the representation to the abstraction.

Q4 a)
Forget to leave 'other' in an empty state.
Original data is not deleted (memory leak).

Q5 a)
• Some students didn't include base classes for subject/observer and instead directly put notify/update methods in the Race/Bettor classes.
• No 'addObserver' and 'removeObserver' in Subject. Or made them private.
• Subject has an aggregation of Observer.
• A few students made the subject/observer relationship between the Bet and Bettor classes.
• Many students forget to make the base classes abstract and the update method virtual.
• A few students got the wrong public/private/protected status for notify/update.

Q5 b)
Most students did well on this question. Common mistakes were:
• subtotal variable in printBill declared as an integer rather than a double or a float,
• not using Date::today() to obtain the current date
• forgetting that minNumDays() and maxNumDays() were functions
• UML in part ii) shows methods in FreeShipping have protected visibility, not public and not private
• forgetting that FreeShipping needs to inherit publicly from Order