17.1 Describe the two parts of a node.

Container data and one or two links to other nodes.

17.2 What is a list head?

A reference to a node which is not referenced by another node (or which is referenced by the tail node, in the case of circular lists, or by the next node, in the case of doubly-linked lists).

17.3 What signifies the end of a linked list?

A node with a NULL reference for the next node.

17.4 What is a self-referential data structure?

A data structure which holds a pointer to another data structure as the same type as itself.

17.5 What is the difference between appending a node to a list and inserting a node into a list?

Appending a node does not require modifying the appended container.

17.6 Which is easier to code: appending or inserting?

Appending (though not by much).

17.7 Why does the `insertNode` function shown in this section use a `previousNode` pointer?

So it can link that node to the inserted node.

17.8 What are the two steps involved in deleting a node from a linked list?

Unlink the node from the list (and re-link the affected nodes around it), and then deallocate the memory for the node that was unlinked.

17.9 When deleting a node, why can't you just use the delete operator to remove it from memory? Why must you take the steps you listed in response to Question 17.8?

If you imagine the linked list like a bridge over the mouth of a giant active volcano, if you unlink a node without re-linking the surrounding nodes back together, you create a hole in which cars, busses, tanks, and small children will fall to their doom. Gollum and the Ring-style. The rest of the bridge is inaccessible, and actually leaked memory because there's no way to get back to it.

17.10 In a program that uses several linked lists, what might eventually happen if the class destructor does not destroy its linked list?

It will leak more and more memory until the heap fills up with abandoned nodes. These nodes will turn to drug abuse until your computer's memory looks like a dystopian Democrat-ruled city, not unlike Detroit.