8.1

line 7: lvalue
line 8: lvalue

line 17: rvalue
line 18: lvalue

line 20: lvalue
line 21: lvalue
line 22: lvalue

line 26: lvalue

line 28: lvalue
line 29: rvalue

line 32: rvalue
line 33: lvalue

8.9

line 67: 2 temporaries

- *Temporary produced on line 53. It is required because the return value of the operator+ member function is an expression, and thus must be evaluated and stored in a temporary object in order to be returned.*
- *Temporary produced on line 67. It is required because the default assignment operator requires an object to assign values from, so the expression x + y gets stored in a temporary object and then that object is used as the basis for assignment.*

line 68: 2 temporaries

- *Temporary created on line 53. It is needed because the return value of the operator+ member function is an expression, and thus must be evaluated and stored in a temporary object in order to be returned.*
- *Temporary produced on line 68. It is required because the default assignment operator requires an object to assign values from, so the expression z + z gets stored in a temporary object and then that object is used as the basis for assignment.*

line 69: 0 temporaries

- *The prefix increment operator returns a reference to a counter, which is a valid argument for the default assignment operator.*

line 70: 1 temporary

- *Temporary created on line 28. This temporary is required in order to return x by value since x is an lvalue.*

line 71: 0 temporaries
- *z is a valid argument to provide to the default assignment operator and requires no temporary to be constructed.*

8.12
line 49: copy construction required
line 50: copy construction required
line 51: move construction required
line 52: move construction required
line 53: copy construction must be elided
line 54: move construction allowed but could be elided
line 55: copy assignment required
line 56: move assignment required
line 57: move assignment required, copy construction must be elided
line 58: move assignment required, copy construction allowed but could be elided
line 59: Widget copy construction required, int copy construction must be elided
line 60: Widget move construction required, int copy construction must be elided
line 61: Widget copy construction must be elided, int copy construction must be elided
line 62: Widget move construction required, int copy construction must be elided

8.28
The main advantage of an array-based stack is that it requires effectively no overhead to implement since elements are stored contiguously in memory. This comes with the inherent disadvantage of needing an arbitrary amount of contiguous memory to be allocated. This means that if the maximum size of the array is reached, it is expensive to resize the array to be larger as it necessitates copying all contents of the array to another location. The main advantage of a node-based stack is that it can grow and shrink on demand with effectively no overhead, since nodes can be placed anywhere in memory. This comes with the inherent disadvantage of needing each node to track the location of at least one other node (depending on the implementation -- various possibilities exist). In summary, node-based stacks required more memory per element, but guarantee a fixed duration for push and pop operations; on the other hand, array-based stacks require minimal memory per elements, but cannot guarantee a fixed duration for push operations, particularly in the case where the stack capacity is exceeded and a new, larger stack array must be allocated.