8.1

// x?

lvalue

// static_caststd::vector<int&&>(x)?

rvalue-xvalue

// x.begin()?

lvalue

// ++i?

lvalue

// i?

lvalue

// *i?

lvalue

// *i += 5?

lvalue

// x[0]?

lvalue

// ++a?

lvalue

// a++?

rvalue-prvalue

// func1(x)?

rvalue-xvalue

8.9

z = x + y; / *???* /

one temporary counter object generated for the function return at line 53 and then copy to z

z = z + z; / *???* /

one temporary counter object generated for the function return at line 53 and then copy to z

y = ++z; / *???* /

no temporary object created

z = y++; / *???* /

one temporary counter object generated for the function return at line 28 and then copy to z

x = z; / *???* /

no temporary object created

8.12

Widget b(a); // ???

copy operation

Widget c = a; // ???

copy operation

Widget d(std::move(c)); // ???

move operation, copy elision not allowed

Widget e = std::move(d); // ???

move operation, copy elision not allowed

Widget f(make_widget_1()); // ???

copy operation, copy elision allowed but likely impossible

Widget g(make_widget_2(true)); // ???

move operation, copy elision allowed but likely impossible

c = a; // ???

copy operation

b = std::move(c); // ???

move operation

a = make_widget_1(); // ???

copy operation, copy elision allowed but likely impossible

a = make_widget_2(true); // ???

move operation, copy elision allowed but likely impossiblev

func_1(a); // ???

copy operation

func_1(std::move(a)); // ???

move operation, copy elision not allowed

func_1(make_widget_1()); // ???

copy operation, copy elision allowed but likely impossible

func_2(std::move(b)); // ???

move operation

8.28

Array-based	Node-based
stored in contiguous location	not stored in contiguous location
fixed in size	dynamic in size
element references are invalidated if array rellocated	element references are stable
element access easily	requires traversal the whole linked list
insertion and deletion take time	insertion and deletion are faster
no per-element storage overhead	per-element storage overhead
requires less memory	requires more space

For node-based implementation, random access is not allowed. If you require constant time insertions/deletions and do not know how many items will be stored in the list. Node-based would be better.

Array-based implementation allows random access. If you know the number of elements in advance and concern about the memory, you can choose Array-based implementation.