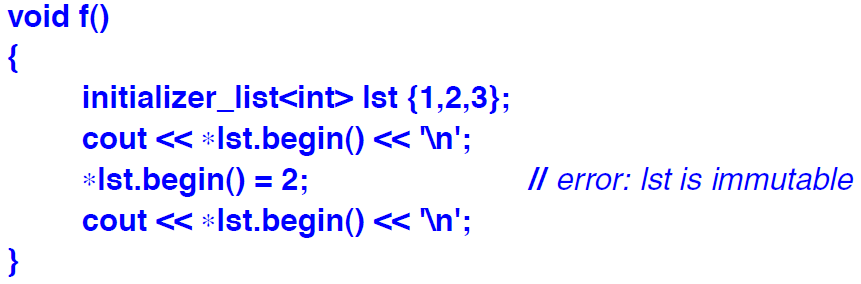
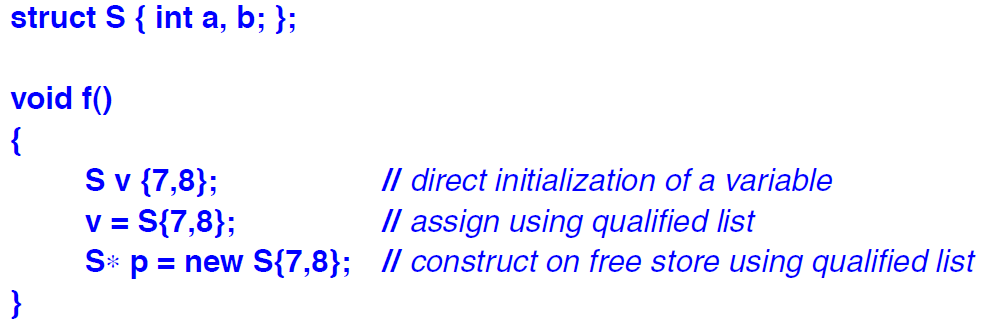
* **Logical Operators –**
* &&(and), ||(or), and !(not) take operands of arithmetic and pointer types.
* && and || operators evaluate their operands only if necessary.
* **Bitwise Logical Operators –**
* &(and), |(or), ^ (exclusive-or / xor), ~ (complement), >> (right shift), << (left shift) are applied to integral types.
* Integral types are char, short, int, long, long long and their unsigned counterparts, and bool, wchar\_t, char16\_t, char32\_t.
* A plain enum can be implicitly converted to an integer type and used as an operand to bitwise logical operations.
* **Conditional Expressions –**
* Conditional expressions are important in that they can be used in constant expressions.
* **Free Store –**
* A C++ implementation does not guarantee the presence of a garbage collector that looks out for unreferenced objects.
* Consequently, objects created by *new* are manually freed using *delete*.
* The operation *delete* can be applied only to a pointer returned by *new* or to a *nullptr*.
* Applying *delete* to *nullptr* has no effect.
* If the deleted object is of a class with a destructor, that destructor is called by *delete* before the object’s memory is released for reuse.
* **Memory Management –**
* The main problems with free store are –
* ***Leaked Objects:*** People use new and then forget to delete the allocated object. This can cause a program to run out of space.
* ***Premature Deletion:*** People delete an object that they have some other pointer to and then later use that other pointer. The pointer to the deleted object no longer points to a valid object. Reading it may give bad results. Writing to it may corrupt an unrelated object.
* ***Double Deletion:*** An object is deleted twice, invoking its destructor (if any) twice. By the second delete, the memory pointed to by another object may have been deleted, causing disastrous results.
* There are two approaches to resource management to avoid the problems of free store –
* Prefer scoped variables to free store objects.
* Use standard-library containers such as string, vector, unique\_ptr, shared\_ptr. This rule is referred to as RAII (Resource Acquisition Is Initialisation).
* **Getting memory space –**
* The free store operators *new*, *delete*, *new[]*, *delete[]* have their definitions in the *<new>* header.
* **List Implementation Model –**
* Having a {}-list be immutable implies that the container taking elements from it must use a copy operation rather than a move operation.



* The lifetime of a {}-list is determined by the scope in which it is used.
* **Qualified Lists –**



* A {}-list is the simplest way of dealing with homogenous lists of varying lengths.