1. If you put **shared\_ptrs** in a container, and you subsequently need to use some, but not all, of the elements, remember to **erase** the elements you no longer need.
2. One common reason to use dynamic memory is to allow multiple objects to share the same state.
3. By default, dynamically allocated objects are default initialized, which means that objects of built-in or compound type have undefined value;
4. We can also **value initialize** a dynamically allocated obj by following the type name with a pair of empty parentheses.
5. For the same reasons as we usually initialize variables, it is also a good idea to initialize dynamically allocated objects.
6. When we provide an initializer inside parentheses, we can use **auto** to deduce the type of the object we want to allocate from that initializer. [Example](Dynamic%20Memory.cpp)
7. It is legal to use new to allocate **const** obj. [Example](Dynamic%20Memory.cpp)
8. By default, if **new** is unable to allocate the requested storage, it throws an exception of type **bad\_alloc**, we can prevent **new** from throwing an exception by using placement new and pass **nothrow** to it. [Example](Dynamic%20Memory.cpp)
9. Making the pointer null after delete can sometimes detect the error using an obj after it has been deleted
10. A **shared\_ptr** can coordinate destruction only with other **shared\_ptrs** that are copies of itself. Indeed, this fact is one of the reasons we recommend using **make\_shared** rather than **new**.
11. When we bind a **shared\_ptr** to a plain pointer, we give responsibility for that memory to that **shared\_ptr**. Once we give **shared\_ptr** responsibility for a pointer, we should no longer use built-in pointer to access the memory to which **shared\_ptr** now points.
12. Use **get** only to pass access to the pointer to code that you know will not **delete** the pointer. In particular, never use **get** to initialize or assign to another smart pointer.
13. If you use a smart pointer to manage a resource other than memory allocated by new, remember to pass a **deleter**.
14. Most applications should use a library container rather than dynamically allocated arrays. Using a container is easier, less likely to contain memory-management bugs, and is likely to give better performance.
15. It is important to remember that what we call a dynamic array does not have an array type.
16. We can value init the elements in an array by following the size with an empty pair of parentheses.
17. Under the new standard(C++11), we can also provide a braced list of element initializers.
18. Although we can use empty parentheses to value init the elements of an array, we cannot supply an element initializer inside the parentheses, which means that we cannot use **auto** to allocate an array.
19. It is legal to dynamically allocate an empty array. you can use the return pointer like an end () iterator (you cannot dereference).
20. for delete [], elements in an array are destroyed in reverse order.
21. We must construct objects in order to use memory returned by allocate. Using unconstructed memory in other ways is undefined.