* **MACROS** : Why are functions being declared as macros? In some cases, it’s useful to avoid function call overhead. Something like inline, but inline is a request to the compiler that could be ignored. Macro functions are expanded by the pre-processor so in a way forces the inline aspect.
* #ifdef and #endif are called compile-time flags. Stuff like this comes under compile-time infrastructure. Look at fluentcpp article by Foster Brereton. In it, the idea is to **get meaningful compiler errors** when using #if and #ifdef macros and the usefulness of function-like macros. Function-like macros require definition when they are used.
* **Unique\_ptr :** Note that a unique pointer manages the ownership of the pointer. It does not prevent us from accessing the raw ptr. That’s why we have the unique\_ptr::get() function. Its ok to do:

**Unique\_ptr<int> uptr(new int);**

**Int\* p = uptr.get();**

**But don’t attempt to delete the memory resource using p.**