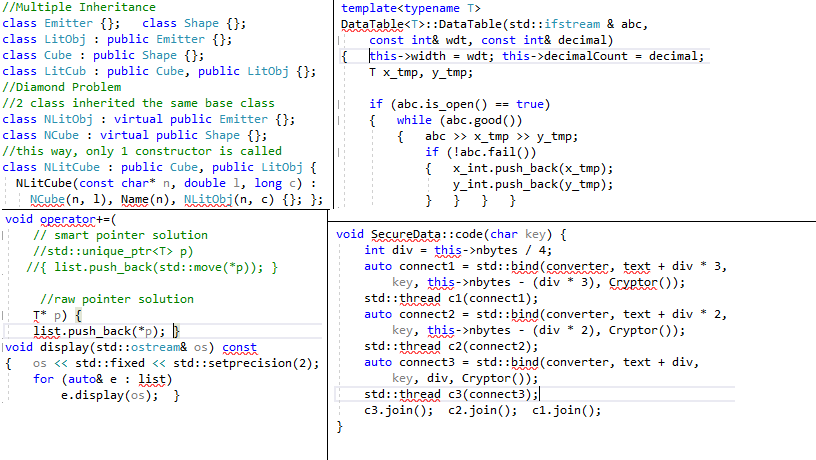
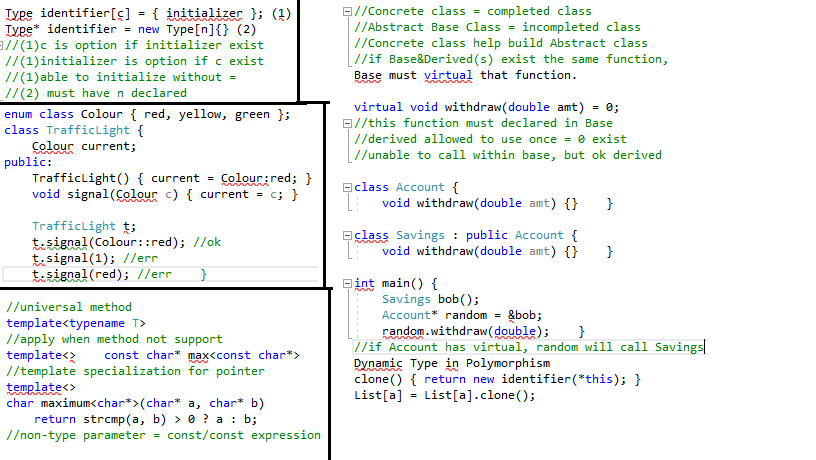
**Type** (fundamental = variable, built-in, compound = class): specify data type to be stored, allocated fix amount of memory and operation to the data type. **Linkage**: refer to same entity across process: external (extern-connect var across scope in different modules), internal(static) and non-linkage (not avail) **Universal Declare**: int x {1} or double y {1.1} **Pointer**: hold an address of value, difference between type/type\*, require casting (static\_cast<type>) to assign pointer to another pointer **lValue(&)**: has declaration, modifiable value(int x = 3/x++/int y = x/)(std::ref() return lValue) **rValue(&&)**: temporary object (a num 2/\*p = 1/q=p+5/)(std::move() return rValue) **Iterator**: for(auto &e:(var value)) **prValue**: any single number(include ++/--) **Class**: Class default private, Struct/Union default public **Shallow Copy**: copy value directly without considering anything **Deep Copy**: copy one by one, allocate memory where necessary **Extern**: borrow the file scope variable declaration from the main() to the current file (borrow int i = 0 in main(), declare extern int i) **Static**: redeclare variable that has already exist somewhere else (has to be file scope and especially in main()) **Copy Constructors**: copy current object to newly created object with the same type **Assignment Operator**: defines logic of copying data from constructor **Move Operators**: use rvalues and make shallow copy (only value) over current object **Inheritance**: Derived Class: status Base Class (Base own Derived) **Template**: support multi-type argument data type **Compositions Class**: contains another class, responsible for copying and destroying the resources.Object use gets affected by destructor **Aggregations Class**: create a new instance and partially modify it. The object used doesn’t get the effect by destructor doesn’t get the effect. Object destroyed in both way if one is triggered **Association Class**: doesn’t involve ownership of any type. The object gets destroy in one way and exist the other way **x++** is accessing the value before increment it, won’t work for lValue **++x** is increment the value before accessing it, work for lValue **Function Object**: where the object has been overloaded. It can be passed to any functions because it has a type(passing Function Object as an argument) **Error Handling(try/throw/catch):** if a throw is found during executing try, it will skip rest of the try and catch the corresponding error from the throw. When the catch executes, it immediately transfers the control back to the system. Catch error from Class Derived first before getting to std::exception. **noexcept**: Terminate program immediately if “catch” throw without try **Normal Exits:** require function calling for exit (ext()), safely destroy the object, thread,flushes and closed all C stream **Abnormal exit:** exit the program right away without any further instruction (terminate(), abort()) **Raw Pointer:** direct access to memory, unable to modify, (+/-) is used to move the pos of element in array, user responsible to deallocate memory. If exception handling exist, raw pointer must deallocate memory anywhere where a return/throw exist **Smart Pointer:** manage memory it's pointing to, auto dereferencing, auto-initialized and deallocating memory **Exclusive Memory(Unique):** one object own resource at the time, call destructor when the object gets destroyed, cannot copy (but can move) to another pointer, provide exception handling  **Exclusive Memory(Shared):** many objects can refer to the object or resources, can copy and move, can only destroy itself when no others smart pointer is referenced to it (destroy other smart pointers first before destroying the main one), initialized, destroy and exception like shared **Task**: group of instruction **Process**: program executing (required resources, move various states) **Multitasking Scheduler**: manages process access to resources **Thread**: separate the process work into mini-process, create new concurrent task. If threads exist in the array, the order can be mixed **Race Condition**: when 2 threads trying to access the same block of memory. This could lead to different output everytime the program is run. Shared states, mutexes, lock, and atomics can help. **Deadlocks**: two or more threads wait for another thread to complete execution and get blocked forever **Shared States**: shared states of 2 threads to communicate about the same reference they are updating **Mutexes/Locks**: Mutexes (Mutual Exclusion) prevents other threads to update/use the value while a thread is using. Mutexes will temporarily lock the variable and release once the thread is done its job **Future**: an object received its value through shared-state once it’s ready, usually through a provider. Provider “promise” future will get the value from shared-state **Preprocessor**: prepare a translation unit and resolves all directives in source code (#) **Multiple Inheritance**: allow a class to inherit the class of more than 1 base class **Diamond Problem**: when 2 class inherited the same base class (replicated base class), virtual must be used to eliminate ambiguity **Inheritance**: sub-class benefits from base class where it can reuse the class structure



\*\*Note\*\*: Difference between aggregation and association is aggregation is aggregation can only remove objects from the list while aggregation can assign the object to another object, leave nothing get destroy. \*\*Note\*\*: Move constructor/assignment will do the work faster than Copy constructor/assignment since it doesn’t do any type check, allocation or deallocation \*\*Note\*\*: if Base class exist virtual function, all Child class must use that function and if a child doesn’t use it, make it virtual … = 0; otherwise **\*\*Note\*\*:** Since Smart Pointer automatically handling the exception, when the first Smart Pointer execute and detect an exception, it will ignore the entire process after it, return control to main()

