This note is a set of tips and tricks that I have learnt in coding with the C++ programming language.

RAII

RAII stands for *Resource Acquisition Is Initialization*. It is officially defined as “a C++ programming technique which binds the life cycle of a resource that must be acquired before use to the lifetime of an object. RAII guarantees that the resource is available to any function that may access the object. It also guarantees that all resources are released when the lifetime of their controlling object ends.” This briefly means that we need to:

1. Encapsulate each resource into a class: the constructor will acquire the resource, and the destructor releases the resource
2. Use the resource via a local instance of the class (i.e., not a pointer or static variable)

Let look at the following example.

Although function `notUseRAII` looks good, there is a potential memory leak. If an exception occurs while `doing something`, then `delete` is not invoked. Thus, there will be a memory leak.

Now, let apply RAII by wrapping `msg` using a `struct`.

As `wrapper` is a local variable, its destructor is guaranteed to be invoked when leaving function `useRAII` for any reason. Thus, the memory of the `msg` is ensured to be deallocated.

Ref: <https://en.cppreference.com/w/cpp/language/raii>

Working with containers

1. `auto` specifier in range-based for loop

Range-based for loop has been introduced since C++11 and used as a more readable equivalent to the traditional for loop operating. The `range\_declaration` part often use the `auto` specifier for automatic type deduction (and reduce code modification when updating elements' type.) It is important to remember that `auto` represents a value type. Thus, each iteration of the loop copies an element of the container, and any modification won't affect the corresponding original member. Let see the following example; method `append` does not modify any strings inside `my\_strings`.

To make the effect, we need to modify the `auto` with `&` symbol to represent a reference type. As denoted below:

1. Finding an element with std::find\_if

A class or a struct:

When moving from C to C++, I often ask “should I use a `struct` or a `class`?”

In term of language, they are almost equivalent. The only difference is their default visibility (i.e., `public`, `private`, or `protected`) of members and inheritance. (It is `public` in `struct` and `private` in `class`.) And this does not help to answer the question.

Fortunately, there is

Static analysis tools