Week10

a. How does keyword auto work in C++?

`auto` 关键字在 C++ 中用于自动类型推导(deduce)。当您使用 `auto` 声明一个变量并初始化它时，编译器会检查初始化表达式的类型，并自动确定(work out)该变量应该具有的正确类型。

---

b. Why you shouldn't use auto too much?

不应该过度使用 `auto` 的主要原因是它会降低代码的可读性(less readable)。当其他人阅读代码时，如果只看到 `auto`，他们无法立即知道变量的确切类型。这迫使他们必须依赖 IDE 的功能(如鼠标悬停提示)或去查找返回该变量的函数定义，这会花费更多时间。此外，如果一个 API 的返回类型发生了变化，`auto` 可能会隐藏这个变化，导致后续依赖该类型的代码以不易察觉的方式出错。

---

c. When it's a good idea to use auto?

当变量的类型名称非常长(lengthy/huge)时，使用 `auto` 是一个好主意。这主要有两种情况：

1. **迭代器 (Iterators):** 容器(如 `std::vector`)的迭代器类型声明非常长(例如 `std::vector<std::string>::iterator`)。在这种情况下，使用 `auto` 可以使代码更简洁易读。
2. **复杂的返回类型:** 当函数返回一个非常复杂的类型时，比如一个嵌套的 `std::unordered_map`，其类型声明会非常冗长。使用 `auto` 可以大大缩短代码。

---

d. Describe the use of the keyword using.

`using` 关键字(或 `typedef`)可以用来为非常长的类型名称创建别名(alias)。这可以作为使用 `auto` 的一种替代方案，既能简化代码，又能保持类型的明确性。例如，您可以将一个复杂的 `std::unordered_map` 类型 `using` 为一个更短的名称，如 `DeviceMap` 。

---

e. What's the difference between auto and auto&?

`auto` 和 `auto&`(`auto` 引用)的主要区别在于是否创建副本(copy)。

- **`auto`**: 会创建变量的副本。
- **`auto&`**: 不会创建副本，而是将变量存储为引用(reference)。

---

f. Explain "just because you might be returning a reference, doesn't mean it's actually going to store it as a reference".

这句话的意思是：即使一个函数返回的是一个引用，如果你使用 `auto` 关键字(不带 `&`)来接收这个返回值，C++ 也不会自动将其存储为引用。相反，它仍然会复制(copy)这个变量。你必须显式地使用 `auto&`(视频中提到的是 `const auto reference`)来声明接收变量，才能确保它被存储为引用，从而避免复制。

---

g. In `void (\*function)() = HelloWorld`, identify the type, the name, and the initial value of the function pointer, respectively.

根据视频中 `void (*function)() = hello world` 的示例(以及后来明确写出的类型 `void (*function)()`：

- **Type (类型):** `void (*)()` 。这是一个指向函数的指针，该函数不接受任何参数且返回 `void`。
- **Name (名称):** `function` 20。
- **Initial Value (初始值):** `HelloWorld`。这是 `HelloWorld` 函数的内存地址。

---

h. Describe the use of the keyword typedef.

`typedef` 关键字用于为现有的类型创建一个别名(alias)，这对于复杂的类型特别有用，尤其是函数指针。由于原始的函数指针语法(如 `void (*MyFunction)()`)非常混乱(confusing)且难以阅读，使用 `typedef` 可以定义一个更清晰、更易读的名称(例如 `HelloWorldFunction`)，然后使用这个别名来声明变量。

---

i. What is the different between a lambda expression and a regular function?

Lambda 表达式是一种定义“匿名函数”(anonymous function)的方式。它与常规(regular)函数的主要区别在于：

1. **声明方式:** 常规函数需要一个正式的函数定义(formal function definition)。Lambda 是一种“快速可丢弃”(quick disposable)的函数，它没有像常规函数那样的正式声明，而是在使用时“内联”(in line)定义。
2. **匿名性:** Lambda 是匿名的，它没有函数名(除非被赋值给一个变量)。
3. **本质:** Lambda 更像一个可以被传递的变量，而常规函数是在编译代码中存在的一个正式符号(symbol)。