# Week8

**a. Why is the console almost guaranteed to work?**

控制台几乎可以保证工作，主要是因为它基本上是内置于操作系统中的。不像图形显示系统可能会因为图形渲染系统的问题而无法正常显示信息，控制台作为一个信息转储(information dump)，其运行不依赖于应用程序中可能出错的图形系统，因此非常可靠，尤其适用于调试目的。

**b. What does the m_ prefix convention imply for a member variable?**

`m_`(或 `m` 下划线)前缀是一个命名约定，用来表明一个变量是类的私有成员变量。这种约定有助于在代码中区分成员变量和局部变量，从而提高代码的可读性和组织性，尤其是在处理大型代码库和复杂类时。

**c. Can you use multiple public or private keywords in the definition of a class?**

是的，可以在一个类的定义中使用多个 public 或 private(以及 protected)关键。这允许程序员根据自己的风格将类的不同部分(例如公共方法和公共变量)分开，以提高代码的组织性和可读性。这是一种个人风格的选择，并非语言强制要求。

**d. When should you use an enum type?**

当你想用整数来表示某些特定的状态或值，并且希望给这些值赋予有意义的名字以提高代码的可读性时，就应该使用枚举(enum)类型。它有助于将一组相关的常量值组织在一起，并可以(在一定程度上)限制变量只能取枚举中定义的值，而不是任意整数。

**e. What is the underlying data type of an enum type?**

枚举(enum)的底层数据类型是整数。默认情况下，它通常是 int(一般为 32 位)。但是，你可以通过在枚举名称后面加上冒号和指定的整数类型(如 unsigned char)来显式指定一个不同的整数类型作为底层类型，以便在值范围允许的情况下节省内存。不能使用 float 等非整型作为底层类型。

**f. What is the difference between the enum type and any integer type?**

虽然枚举本质上是整数，但使用枚举类型相比于直接使用整数类型(如 int)提供了更强的类型检查和语义含义。将一个变量声明为枚举类型意味着它的意图是只持有该枚举定义的一组命名值，而普通的整数变量可以持有任何整数值。这有助于限制变量的取值范围(编译器层面的限制)，并通过使用有意义的名称而不是“魔法数字”(magic numbers)来提高代码的可读性。

**g. Can a function and a variable share a name within the same namespace? What about in different namespaces?**

- **在同一命名空间/作用域内(例如类内部)：** 通常不可以。如果一个函数和一个变量(或枚举常量)共享相同的名称，会导致歧义，编译器无法确定名称指的是哪一个，从而产生编译错误。
- **在不同命名空间内：** 可以。命名空间的设计目的之一就是为了解决命名冲突。不同命名空间中的同名函数和变量可以通过命名空间限定符(如 `NamespaceA::name` 和 `NamespaceB::name`)来区分。

**h. What is the primary use of a constructor?**

构造函数(constructor)的主要用途是在创建类的一个新实例(对象)时对其进行初始化。这包括为成员变量设置初始值(特别是 C++ 中不会自动初始化的基本类型成员)，以及获取对象运行所需的资源(如动态分配内存)。

**i. How does a constructor differ from another method performing the same task?**

构造函数是一种特殊的类方法，它不同于用于执行类似初始化任务的普通成员函数(例如一个名为 Init 的方法)：

1. **自动调用：** 构造函数在对象创建时由编译器自动调用。开发者不需要像调用普通方法那样显式调用它。
2. **命名规则：** 构造函数的名称必须与类的名称完全相同。
3. **无返回类型：** 构造函数没有返回类型，连 `void` 都没有。普通成员函数必须指定返回类型。

### j. What does the C++ default constructor do?

如果你没有为类定义任何构造函数，C++ 编译器会自动提供一个默认构造函数。这个默认构造函数对于类的内置类型成员(如 int, float, 指针)不做任何初始化，这些成员将含有不确定的“垃圾”值。然而，如果类包含其他类的对象作为成员变量，默认构造函数会调用这些成员对象的默认构造函数。

### k. When will the destructor be called?

析构函数(destructor)在对象的生命周期结束，即将被销毁时自动调用。这主要发生在两种情况下：

1. **栈对象：** 当在栈上创建的对象离开其作用域时(例如，执行到其声明所在的代码块的末尾 `}` 时)，其析构函数会被调用。
2. **堆对象：** 当对一个指向在堆上动态分配(使用 `new`)的对象的指针使用 `delete` 操作符时，该对象的析构函数会在内存被释放之前调用。

### l. What is the primary use of a destructor?

析构函数的主要用途是释放对象在其生命周期内获取的资源。这通常是为了执行必要的清理工作，以防止资源泄漏。常见的任务包括：释放通过 new 动态分配的内存、关闭文件句柄、断开网络连接、解锁互斥锁等。