Week9

a. What does the name of an array stand for and what is its type?

数组的名称(例如 `example`)本身代表其内存地址。它本质上是一个指针，其类型取决于数组存储的元素类型。例如，一个 `int` 类型的数组，其名称就是一个整数指针(integer pointer)，指向该数组在内存中的起始地址。

b. How are the elements of an array stored in memory?

数组的元素在内存中是**连续存储的**(contiguously)，这意味着它们的数据在内存中是挨在一起的(in a row)。例如，一个包含5个整数的数组，会在内存中连续分配20个字节(假设整数为4字节)，一个整数紧接着另一个整数存放。

c. What is pointer arithmetic?

指针算术(pointer arithmetic)指的是对指针进行算术运算，例如给指针加一个整数值。这种运算增加的实际字节数(bytes)取决于指针的类型。例如，如果一个整数指针(integer pointer)加上 `2`，它实际增加的地址偏移量是 2 乘以一个整数的大小(例如 2 * 4 = 8 字节)，从而使其指向数组中索引为2的元素。

d. What is the difference between an array allocated on the stack and an array allocated on the heap?

它们最主要的区别在于**生命周期**(lifetime)。

- **栈 (Stack) 数组:** 在栈上创建的数组，其生命周期局限于当前的作用域(scope)。当程序执行离开该作用域(例如遇到反大括号 `}`)时，它占用的内存会被自动销毁(destroyed)。
- **堆 (Heap) 数组:** 在堆上创建的数组(使用 `new` 关键字)，它会一直存在(alive)，直到程序结束，或者直到你手动使用 `delete` 关键字(并且因为是数组，需要使用 `delete[]`)将其显式销毁。

e. What do you need to pay special attention to when using raw arrays?

使用原始数组(raw arrays)时，必须特别注意以下几点：

1. **边界检查 (Bounds Checking):** 必须确保你始终在数组的边界(bounds)内进行读写。尝试访问无效索引(例如-1或超出分配大小的索引)会导致内存访问冲突(memory access violation)。
2. **内存覆盖 (Memory Corruption):** 写入数组边界之外的内存，可能会无意中修改程序中其他变量的值，这会导致非常难以调试(debug)的问题。因此，需要设置安全检查(safety checks)来防止越界写入。
3. **大小管理 (Size Management):** C++ 无法自动获知一个原始数组的大小(特别是从堆分配的数组)。程序员必须自己来维护(maintain)数组的大小信息。

f. When you create an array on the stack, its size must be known at compile time. What about the heap?

根据提供的资料，在栈(stack)上分配数组时，其大小必须是一个在编译时(compile time)已知的常量。

关于在堆(heap)上分配数组，资料中展示了使用 `new` 关键字(例如 `new int[5]`)的示例，但**并未明确说明**其大小是否必须在编译时确定。

g. A good goal for software engineering is to avoid code duplication. What is the benefit of this achieving goal?

避免代码重复(code duplication)的主要好处在于**可维护性**。

如果您决定更改某项功能的实现逻辑(例如，将输出方式从 `cout` 更改为 `printf` 或其他自定义日志函数)，您只需要在一个地方修改代码。如果存在多份重复的代码，您就必须在所有定义了该功能的地方都进行修改。

h. What is a template? Is it a real function or a real class?

模板(template)**不是**一个真实的函数或真实的类。

它更像是一个“蓝图”(blueprint)或一组规则。你通过模板来告诉编译器(compiler)如何根据你的规则为你自动编写(write code for you)和生成代码。

i. When will a function or a class be created from its template?

模板本身并不会被编译成代码。它只在程序中**实际调用(call)或使用(usage)它**时，才会真正被创建(created)或“实例化”(materialized)为真实的、可编译的源代码。编译器会根据你使用模板时提供的参数(例如特定的类型或值)来生成具体的函数或类。

j. Is it okay to invoke a template function without specifying the type in <>?

是的，可以。

如果您不显式地在尖括号(angular brackets)中指定类型，编译器会尝试从传递给函数的参数中**自动推导**(automatically deduce)所需的类型。例如，当你传递 `5` 作为参数时，C++ 知道 `5` 是一个整数(integer)，因此可以自动推导出模板类型T应该是 `int` 。