Week7

a. A string is basically an array of what?

字符串(string)基本上是一个字符(characters)数组。

b. What is a C style string?

C风格字符串(C style string)是一种定义字符串的方式，例如，通过 const char 指针(const char pointer)来声明，并将其设置为用双引号括起来的文本。

c. What indicates the end of a string?

字符串的结尾由一个设置为零的字节来指示，这个字节被称为空终止字符(null termination character)。当程序(例如 std::cout)处理字符串时，它会一直运行，直到遇到这个零值，此时它才意识到这是字符串的结尾。

d. What is the difference between 0, '0', and "0"?

根据提供的文本：

- `0`：是一个数字常量(numeric constant)，可以用来作为字符串的空终止符。
- `'0'`：文本没有直接定义 `'0'`，但指出字符(characters)是使用单引号(single quotes)定义的。
- `"0"`：文本指出，使用双引号(double quotes)定义的内容默认会成为一个 `char` 指针(char pointer)。

e. What is the relation between std::string and std::basic_string?

`std::basic_string` 是一个模板类(template class)。`std::string` 是 `std::basic_string` 类的一个模板特化(template specialization)，它使用 `char` 作为模板参数(template parameter)。

f. What is the best way to pass a string into a read-only function?

将字符串传递给只读(read-only)函数时，最好的方式是“通过常量引用(const reference)传递”。这包含两层含义：const(常量)意味着你承诺不会在函数内修改该字符串；reference(引用)意味着该字符串不会被复制，从而避免了代价高昂的字符串拷贝操作。

g. What is a typical size of a stack?

栈(stack)通常是一个具有预定义大小(predefined size)的内存区域，其大小“通常在两兆字节(two megabytes)左右”。

h. How are objects close to each other in the program stored in stack memory?

在栈内存中，变量(例如一个整数、一个数组和一个向量对象)在内存中“都非常接近(close together)”，它们“紧挨着(right next to each other)”彼此。内存“就像一个栈一样，逐个堆叠在一起(stored on top of each other like a stack)”。

i. Why are stack allocation and stack deallocation extremely fast?

栈分配(stack allocation)极其快速，因为它“字面上就像一个CPU指令(one CPU instruction)”。当在栈上分配变量时，实际发生的只是栈指针(stack pointer)移动相应数量的字节。栈释放(deallocation)也同样快速，它“基本上是免费的(basically free)”，也只是一个CPU指令的操作，即栈指针移回(pop)到该作用域开始前的位置。

j. What happens when a scope comes to an end?

当一个作用域(scope)结束时，“在该作用域的栈(stack)上分配的所有内存都会被弹出(popped off)”，即被释放(freed)和回收(reclaimed)。

k. Why is heap allocation slow?

堆分配(heap allocation)之所以缓慢，是因为它涉及一个复杂的过程，而栈分配“则像一条CPU指令”。new 关键字会调用 malloc，malloc 必须维护一个“空闲列表(free list)”来跟踪哪些内存块是空闲的。malloc 需要遍历该列表，找到一个足够大的空闲块，并执行“大量的簿记(bookkeeping)工作”来记录此次分配的大小和状态。如果空闲列表中的内存不足，程序还必须向操作系统请求更多内存，这是一个“非常昂贵(very expensive)”的操作。

l. When would you allocate an object on the heap?

在堆(heap)上分配对象的“唯一真正原因(only reason really)”是当对象“无法在栈(stack)上分配”时。这通常发生在两种情况下：

1. 你需要对象的生命周期(lifetime)“比你的函数作用域(scope of your function)更长”。
2. 你需要分配“更多的数据(more data)”，例如加载一个50兆字节(50 megabytes)的纹理，它“无法装入栈中(not gonna fit onto the stack)”。