Week3

关于 C++ 中的变量 （第8集）

- **问题a: C++ 是一门功能强大、规则很少的语言，那么 C++ 中那些基本数据类型之间的本质区别是什么？**
  C++ 是一门规则很少的强大语言。C++ 中不同变量类型之间唯一的本质区别是它的大小，即该变量在内存中占据多大的空间。
- **问题b: `int` 变量可以存储的确切值范围是多少？**
  一个有符号的 `int` 变量通常是4个字节大小，可以存储大约从负20亿到正20亿的数值。这个范围的由来是，4个字节等于32位。因为是有符号的，其中1位用于表示正负号，剩下31位用于表示数值。因此，最大值约为2^{31}，即大约21亿。所以，它可以存储从0到约21亿的正数，以及到约负21亿的负数。
- **问题c: `unsigned` 关键字有什么用？**

  `unsigned` 关键字用于定义一个没有符号的数字，意味着它总是正数。它去掉了用于表示负号的符号位，让该位也用于表示数值。这使得一个整数变量可以使用全部32位来存储数值，从而可以存储一个更大的数字，范围从0到大约42.9亿（2^{32}）。
- **问题d: 还有哪些其他的整型数据类型？**
- 除了 `int`，其他的基本整型数据类型包括 ：
  - `char` ：1个字节 。
  - `short` ：2个字节 。
  - `long` ：通常是4个字节，取决于编译器 。
  - `long long` ：通常是8个字节 。
  - 此外还有像 `long int` 这样的其他类型。
- **问题e: 字符在 C++ 中也只是整数。程序是如何区分它们的？**
  字符本质上就是数字，例如字符'A'的数值是65。程序之所以能区分它们，是因为程序员们对特定的数据类型有惯例和假设。例如，像 `cout` 这样的工具在接收到一个 `char` 类型的变量时，会将其当作一个字符来处理并打印出字符本身；而如果接收到

的是`short`等其他整数类型，即使赋给它一个字符，`cout`也会将其当作一个数字来处理并打印出其数值。

- **问题f: 在一个十进制数末尾追加`f`或`F`有什么用？**
  在一个十进制数（如5.5）的末尾追加`f`或`F`是为了告诉编译器这是一个`float`类型的变量。如果不加`f`，编译器会默认这个十进制数是`double`类型。
- **问题g: 为什么`bool`数据类型占用1个字节的内存，而它实际上只需要1位来表示`true`或`false`？**
  尽管`bool`类型只需要1位来表示`true`或`false`，但它在内存中仍然占用1个字节。这是因为计算机在内存寻址时，无法定位到单个的位（bit），只能以字节（byte）为单位进行寻址和访问。因此，无法创建一个只有1位大小的变量类型，因为我们无法访问它。

关于 C++ 中的函数（第9集）

- **问题h: 对于一个函数来说，返回类型`void`意味着什么？**
  返回类型`void`意味着函数不返回任何东西。这种函数只是执行一个特定的任务，比如将结果打印到控制台，而不会返回一个值。
- **问题i: 当你调用一个函数时，你的程序中会发生什么？请简要描述。**
  当调用一个函数时，编译器会生成一个调用指令。程序需要为该函数创建一个完整的栈帧（stack frame），包括将函数的参数和返回地址等信息推入栈中。然后，程序会跳转到二进制文件的另一部分去执行该函数的指令。执行完毕后，程序需要通过之前存入的返回地址回到调用函数之前的位置。
- **问题j: 为什么为每一行代码都创建一个函数不是一个好主意？**
  为每一行代码都创建一个函数不是一个好主意，因为这会让代码难以维护，看起来混乱不堪，并且会使程序变慢。程序变慢的原因是，每次函数调用都需要执行创建栈帧、跳转内存地址等操作，这些都需要花费时间。

关于 C++ 头文件（第10集）

- **问题k: 函数定义和函数声明之间有什么区别？**
  - **函数声明 (Declaration)**：只是告诉编译器某个函数存在，它包含了函数的签名（返回类型、函数名、参数列表），但没有函数体。声明只是说“嘿，存在这样一个函数”，但没有说明这个函数具体做什么。
  - **函数定义 (Definition)**：包含了完整的函数体，即函数的具体实现，它说明了这个函数实际上做了什么。

- **问题l: 我们可以多次定义一个函数和多次声明一个函数吗？**
  一个函数只能被定义一次。如果一个头文件被意外地多次包含进同一个编译单元，并且其中定义了某个结构体，会导致重定义错误。
- **问题m: `#pragma once` 指令有什么用？**

  `#pragma once` 是一个预处理指令，也被称为头文件卫士（header guard）。它的作用是防止同一个头文件在单个编译单元（即单个.cpp 文件）中被包含多次。这样做可以避免因重复包含而导致的重复定义错误。
- **问题n: 解释头文件卫士 `#ifndef`、`#define` 和 `#endif` 的用法。**
  这是一种传统的头文件卫士实现方式。
  - `#ifndef SYMBOL`：检查名为 `SYMBOL` 的符号是否**未被**定义。
  - 如果该符号未被定义，那么从`#ifndef` 到 `#endif` 之间的所有代码都将被包含进编译过程。
  - `#define SYMBOL`：在检查通过后，立即定义这个符号。
  - 这样，当这个头文件下一次被包含时，`#ifndef` 的检查就会失败，因为该符号已经被定义，从而避免了内容的重复包含。
  - `#endif`：标记条件编译块的结束。
- **问题o: 如何区分C标准库头文件和C++标准库头文件？**
  一个区分方法是看文件名是否带有`.h` 扩展名。C标准库中的头文件通常`.h` 扩展名，而C++标准库的头文件则没有。
- **问题p: 在你的电脑上找到 `iostream` 头文件。**
  在Visual Studio中，可以在代码里的`#include <iostream>` 上右键，然后选择“打开文档”。这会打开`iostream` 这个头文件。接着，在打开的`iostream` 文件标签页上右键，选择“打开所在的文件夹”，就可以找到它在电脑上的实际位置。

关于在 Visual Studio 中调试 C++ （第11集）

- **问题q: 调试涉及哪两项主要活动？**
  调试主要包含两个部分：设置断点（breakpoints）和查看内存（looking at memory）。
- **问题r: 为什么在调试前确保IDE处于调试模式很重要？**
  确保处于调试模式（Debug mode）非常重要，因为在发布模式（Release mode）下，编译器会为了优化而重新排列代码，这可能导致你设置的断点永远不会被触发。

此外，调试模式下编译器会做一些额外的事情来帮助调试，比如将未初始化的栈内存填充为 `cc` 值，以便程序员发现问题。

- **问题s: 黄色箭头（指令指针）表示什么？**
  黄色箭头表示程序当前的指令指针所在位置。它指向即将被执行的那行代码，但该行代码本身尚未被执行。
- **问题t: 解释按钮的用法：Continue, Step Into, Step Over,和Step Out。**
  - **Continue （继续）**：让程序正常执行，直到遇到下一个断点或程序结束。
  - **Step Into （步入）**：如果当前行有函数调用，它会进入该函数的内部，从函数的第一行代码开始执行。
  - **Step Over （步过）**：执行当前行的代码，然后移动到当前函数中的下一行。它不会进入当前行调用的函数内部。
  - **Step Out （步出）**：执行完当前函数的剩余部分，然后返回到调用这个函数的地方。
- **问题u: 在调试模式下调出你程序的内存视图。**
  在Visual Studio的菜单栏中，可以通过点击“调试”→ “窗口”→ “内存”→ “内存1”来打开内存视图窗口。
- **问题v: 在变量前放置一个与号（&）的目的是什么？**
  在调试时，在变量名前面加上一个与号（&），可以获取该变量在内存中的地址。
- **问题w: 为什么在调试模式下你会在内存中看到一堆 `cc` ？**
  在调试模式下，内存中出现的一堆 `cc` 值表示这是未初始化的栈内存。这是编译器特意做的，它用 `cc` 来填充这部分内存，目的是为了帮助程序员在调试时，一旦看到这个值就能意识到这里的变量没有被正确初始化。
- **问题x: 在调试模式下，如何跳转到当前函数的剩余部分？**
  如果你想跳过一个循环或几行代码，直接到达当前函数后面的某个位置，可以在你希望程序暂停的目标代码行上设置一个新的断点。然后点击“继续”（Continue）按钮或按F5，程序就会执行中间的所有代码，直到命中你新设置的那个断点为止。