**1、 在main执行之前和之后执行的代码可能是什么？**

**main函数执行之前**，主要就是初始化系统相关资源：

* 设置栈指针
* 初始化静态static变量和global全局变量，即.data段的内容
* 将未初始化部分的全局变量赋初值：数值型short，int，long等为0，bool为FALSE，指针为NULL等等，即.bss段的内容
* 全局对象初始化，在main之前调用构造函数，这是可能会执行前的一些代码
* 将main函数的参数argc，argv等传递给main函数，然后才真正运行main函数
* \_\_attribute\_\_((constructor))

**main函数执行之后**：

* 全局对象的析构函数会在main函数之后执行；
* 可以用 **atexit** 注册一个函数，它会在main 之后执行;
* \_\_attribute\_\_((destructor))

## 2、结构体内存对齐问题？

##### 内存对齐规则

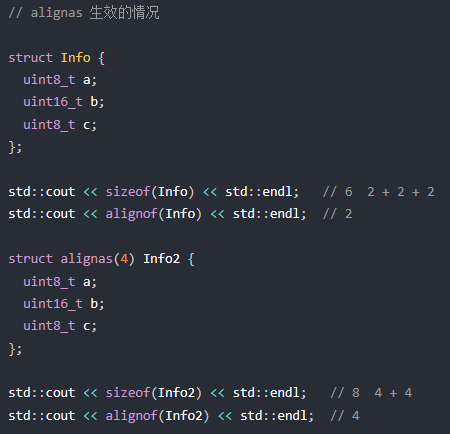
1.第一个成员在与结构体变量偏移量为0的地址处；

2.其它成员变量要对齐到**对齐数**的整数倍地址处。这个对齐数并不是待对齐成员变量的大小，而是**编译器默认对齐数和该成员大小两者中的较小值**。比如V**S中默认的对齐数是8，Linux默认对齐数是4**。

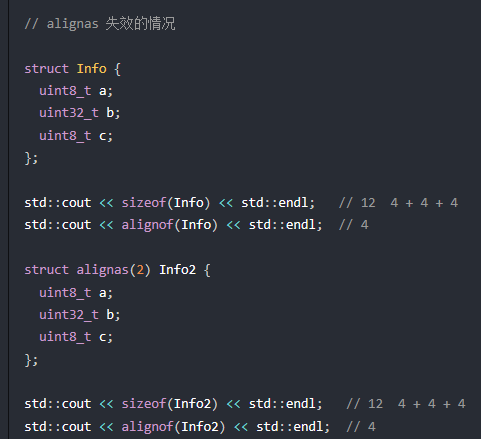
3.结构体整体大小是最大对齐数的整数倍。这个最大对齐数是这样理解的，就是一个结构体里面有多个不同数据类型的成员变量，按照规则2，它们都有一个对齐数，这些对齐数里最大的就是最大对齐数——显然在windows里不会超过8，linux里不会超过4

4.对于结构体里嵌套了结构体的情况，首先里面那个结构体就按照前面三个规则对齐，该内置结构体的首地址就在**自己里面最大对齐数的整数倍处**，比如4的倍数处，对于外面这个结构体，也就是所要求的整个结构体的大小，它就是自己的普通成员变量和嵌套结构体内成员变量中取 最大对齐数 的整数倍。

alignas将内存对齐调整为4个字节。所以sizeof(Info2)的值变为了8：



若alignas小于自然对齐的最小单位，则被忽略：



如果想使用单字节对齐的方式，应该使用#pragma pack(push,1)或者使用\_\_attribute\_\_((packed))

## 3、指针和引用的区别

* 指针是一个变量，存储的是一个地址，引用跟原来的变量实质上是同一个东西，是原变量的别名
* 指针可以有多级，引用只有一级
* 指针可以为空，引用不能为NULL且在定义时必须初始化
* 指针在初始化后可以改变指向，而引用在初始化之后不可再改变
* sizeof指针得到的是本指针的大小，sizeof引用得到的是引用所指向变量的大小
* 当把指针作为参数进行传递时，也是将实参的一个拷贝传递给形参，两者指向的地址相同，但不是同一个变量，在函数中改变这个变量的指向不影响实参，而引用却可以。
* 引用本质是一个指针，同样会占4字节内存；指针是具体变量，需要占用存储空间（32位系统4字节，64位系统8字节）。
* 引用在声明时必须初始化为另一变量，一旦出现必须为typename refname &varname形式；指针声明和定义可以分开，可以先只声明指针变量而不初始化，等用到时再指向具体变量。
* 引用一旦初始化之后就不可以再改变（变量可以被引用为多次，但引用只能作为一个变量引用）；指针变量可以重新指向别的变量。
* 不存在指向空值的引用，必须有具体实体；但是存在指向空值的指针。

**在编译器看来**, int a = 10; int &b = a; 等价于 int \* const b = &a; 而 b = 20; 等价于 \*b = 20; 自动转换为指针和自动解引用

## 4、在传递函数参数时，什么时候该使用指针，什么时候该使用引用呢？

* 需要返回函数内局部变量的内存的时候用指针。使用指针传参需要开辟内存，用完要记得释放指针，不然会内存泄漏。而返回局部变量的引用是没有意义的
* 对栈空间大小比较敏感（比如递归）的时候使用引用。使用引用传递不需要创建临时变量，开销要更小
* 类对象作为参数传递的时候使用引用，这是C++类对象传递的标准方式

## 5、堆和栈的区别

